Run-Time Library (RTL) : Reference guide.

Free Pascal version 3.2.4:
Reference guide for RTL units.
Document version 3.2.4
July 2026

Michaël Van Canneyt

Contents

0.1 Overview . . . 145

1 Reference for unit 'BaseUnix' 146

1.1 Used units . . . 146
1.2 Overview . . . 146
1.3 Constants, types and variables . . . 146
1.3.1 Constants . . . 146
1.3.2 Types . . . 171
1.4 Procedures and functions . . . 184
1.4.1 CreateShellArgV . . . 184
1.4.2 FpAccess . . . 184
1.4.3 FpAlarm . . . 185
1.4.4 FpChdir . . . 186
1.4.5 FpChmod . . . 186
1.4.6 FpChown . . . 187
1.4.7 FpClose . . . 188
1.4.8 FpClosedir . . . 189
1.4.9 FpDup . . . 189
1.4.10 FpDup2 . . . 190
1.4.11 FpExecv . . . 191
1.4.12 FpExecve . . . 192
1.4.13 FpExit . . . 193
1.4.14 FpFcntl . . . 193
1.4.15 fpfdfillset . . . 194
1.4.16 fpFD_CLR . . . 194
1.4.17 fpFD_ISSET . . . 194
1.4.18 fpFD_SET . . . 194
1.4.19 fpFD_ZERO . . . 195
1.4.20 FpFork . . . 195
1.4.21 FPFStat . . . 195
1.4.22 FpFtruncate . . . 196

1.4.23 FpGetcwd . . . 197
1.4.24 FpGetegid . . . 197
1.4.25 FpGetEnv . . . 197
1.4.26 fpgeterrno . . . 198
1.4.27 FpGeteuid . . . 198
1.4.28 FpGetgid . . . 199
1.4.29 FpGetgroups . . . 199
1.4.30 FpGetpgrp . . . 199
1.4.31 FpGetpid . . . 200
1.4.32 FpGetppid . . . 200
1.4.33 fpGetPriority . . . 201
1.4.34 FpGetRLimit . . . 201
1.4.35 FpGetsid . . . 201
1.4.36 FpGetuid . . . 202
1.4.37 FpIOCtl . . . 202
1.4.38 FpKill . . . 203
1.4.39 FpLink . . . 203
1.4.40 FpLseek . . . 204
1.4.41 fpLstat . . . 205
1.4.42 FpMkdir . . . 206
1.4.43 FpMkfifo . . . 207
1.4.44 Fpmmap . . . 207
1.4.45 Fpmprotect . . . 208
1.4.46 Fpmunmap . . . 209
1.4.47 FpNanoSleep . . . 209
1.4.48 fpNice . . . 210
1.4.49 FpOpen . . . 211
1.4.50 FpOpendir . . . 212
1.4.51 FpPause . . . 213
1.4.52 FpPipe . . . 213
1.4.53 FpPoll . . . 214
1.4.54 FppRead . . . 214
1.4.55 FppWrite . . . 215
1.4.56 FpRead . . . 215
1.4.57 FpReaddir . . . 216
1.4.58 fpReadLink . . . 216
1.4.59 FpReadV . . . 218
1.4.60 FpRename . . . 218
1.4.61 FpRmdir . . . 219
1.4.62 fpSelect . . . 219

1.4.63 fpseterrno . . . 220
1.4.64 FpSetgid . . . 221
1.4.65 fpSetPriority . . . 221
1.4.66 FpSetRLimit . . . 221
1.4.67 FpSetsid . . . 222
1.4.68 fpsettimeofday . . . 222
1.4.69 FpSetuid . . . 222
1.4.70 FPSigaction . . . 223
1.4.71 FpSigAddSet . . . 224
1.4.72 FpSigDelSet . . . 224
1.4.73 FpsigEmptySet . . . 224
1.4.74 FpSigFillSet . . . 225
1.4.75 FpSigIsMember . . . 225
1.4.76 FpSignal . . . 225
1.4.77 FpSigPending . . . 226
1.4.78 FpSigProcMask . . . 226
1.4.79 FpSigSuspend . . . 227
1.4.80 FpSigTimedWait . . . 227
1.4.81 FpSleep . . . 227
1.4.82 FpStat . . . 228
1.4.83 fpSymlink . . . 229
1.4.84 fpS_ISBLK . . . 230
1.4.85 fpS_ISCHR . . . 230
1.4.86 fpS_ISDIR . . . 231
1.4.87 fpS_ISFIFO . . . 231
1.4.88 fpS_ISLNK . . . 231
1.4.89 fpS_ISREG . . . 232
1.4.90 fpS_ISSOCK . . . 232
1.4.91 fptime . . . 232
1.4.92 FpTimes . . . 233
1.4.93 FpUmask . . . 233
1.4.94 FpUname . . . 234
1.4.95 FpUnlink . . . 234
1.4.96 FpUtime . . . 234
1.4.97 FpWait . . . 235
1.4.98 FpWaitPid . . . 236
1.4.99 FpWrite . . . 236
1.4.100 FpWriteV . . . 237
1.4.101 FreeShellArgV . . . 237
1.4.102 wexitStatus . . . 237

1.4.103 wifexited . . . 237
1.4.104 wifsignaled . . . 238
1.4.105 wstopsig . . . 238
1.4.106 wtermsig . . . 238
1.5 Dir . . . 238
1.6 Dirent . . . 239
1.7 FLock . . . 239
1.8 iovec . . . 239
1.9 pollfd . . . 239
1.10 sigactionrec . . . 240
1.11 Stat . . . 240
1.12 tfpreg . . . 240
1.13 tfpstate . . . 241
1.14 tfpxreg . . . 241
1.15 tfpx_sw_bytes . . . 242
1.16 timezone . . . 242
1.17 tms . . . 242
1.18 TRLimit . . . 242
1.19 tsigaltstack . . . 243
1.20 TSigContext . . . 243
1.21 tsiginfo . . . 244
1.22 TUcontext . . . 245
1.23 txmmreg . . . 245
1.24 UTimBuf . . . 245
1.25 UtsName . . . 246

2 Reference for unit 'Character' 247
2.1 Used units . . . 247
2.2 Overview . . . 247
2.3 Constants, types and variables . . . 247
2.3.1 Types . . . 247
2.4 Procedures and functions . . . 249
2.4.1 ConvertFromUtf32 . . . 249
2.4.2 ConvertToUtf32 . . . 250
2.4.3 GetNumericValue . . . 250
2.4.4 GetUnicodeCategory . . . 250
2.4.5 IsControl . . . 251
2.4.6 IsDigit . . . 251
2.4.7 IsHighSurrogate . . . 251
2.4.8 IsLetter . . . 251

2.4.9 IsLetterOrDigit . . . 252
2.4.10 IsLower . . . 252
2.4.11 IsLowSurrogate . . . 252
2.4.12 IsNumber . . . 252
2.4.13 IsPunctuation . . . 253
2.4.14 IsSeparator . . . 253
2.4.15 IsSurrogate . . . 253
2.4.16 IsSurrogatePair . . . 253
2.4.17 IsSymbol . . . 254
2.4.18 IsUpper . . . 254
2.4.19 IsWhiteSpace . . . 254
2.4.20 ToLower . . . 254
2.4.21 ToUpper . . . 255
2.5 TCharacter . . . 255
2.5.1 Description . . . 255
2.5.2 Method overview . . . 255
2.5.3 TCharacter.Create . . . 255
2.5.4 TCharacter.ConvertFromUtf32 . . . 256
2.5.5 TCharacter.ConvertToUtf32 . . . 256
2.5.6 TCharacter.GetNumericValue . . . 257
2.5.7 TCharacter.GetUnicodeCategory . . . 257
2.5.8 TCharacter.IsControl . . . 257
2.5.9 TCharacter.IsDigit . . . 258
2.5.10 TCharacter.IsSurrogate . . . 258
2.5.11 TCharacter.IsHighSurrogate . . . 259
2.5.12 TCharacter.IsLowSurrogate . . . 259
2.5.13 TCharacter.IsSurrogatePair . . . 260
2.5.14 TCharacter.IsLetter . . . 260
2.5.15 TCharacter.IsLetterOrDigit . . . 260
2.5.16 TCharacter.IsLower . . . 261
2.5.17 TCharacter.IsNumber . . . 261
2.5.18 TCharacter.IsPunctuation . . . 262
2.5.19 TCharacter.IsSeparator . . . 262
2.5.20 TCharacter.IsSymbol . . . 263
2.5.21 TCharacter.IsUpper . . . 263
2.5.22 TCharacter.IsWhiteSpace . . . 263
2.5.23 TCharacter.ToLower . . . 264
2.5.24 TCharacter.ToUpper . . . 264

3 Reference for unit 'charset' 266

3.1 Used units . . . 266
3.2 Overview . . . 266
3.3 Constants, types and variables . . . 266
3.3.1 Constants . . . 266
3.3.2 Types . . . 266
3.4 Procedures and functions . . . 267
3.4.1 getascii . . . 267
3.4.2 getmap . . . 268
3.4.3 getunicode . . . 268
3.4.4 loadbinaryunicodemapping . . . 268
3.4.5 loadunicodemapping . . . 269
3.4.6 mappingavailable . . . 269
3.4.7 registerbinarymapping . . . 269
3.4.8 registermapping . . . 270
3.5 treversecharmapping . . . 270
3.6 TSerializedMapHeader . . . 270
3.7 tunicodecharmapping . . . 270
3.8 tunicodemap . . . 271

4 Reference for unit 'Classes' 272
4.1 Used units . . . 272
4.2 Overview . . . 272
4.3 Constants, types and variables . . . 272
4.3.1 Constants . . . 272
4.3.2 Types . . . 275
4.3.3 Variables . . . 293
4.4 Procedures and functions . . . 294
4.4.1 ActivateClassGroup . . . 294
4.4.2 BeginGlobalLoading . . . 294
4.4.3 BinToHex . . . 295
4.4.4 Bounds . . . 295
4.4.5 CheckSynchronize . . . 295
4.4.6 ClassGroupOf . . . 296
4.4.7 CollectionsEqual . . . 296
4.4.8 EndGlobalLoading . . . 296
4.4.9 ExtractStrings . . . 296
4.4.10 FindClass . . . 297
4.4.11 FindGlobalComponent . . . 297
4.4.12 FindIdentToInt . . . 297
4.4.13 FindIntToIdent . . . 297

4.4.14 FindNestedComponent . . . 298
4.4.15 GetClass . . . 298
4.4.16 GetFixupInstanceNames . . . 298
4.4.17 GetFixupReferenceNames . . . 298
4.4.18 GlobalFixupReferences . . . 299
4.4.19 GroupDescendentsWith . . . 299
4.4.20 HexToBin . . . 299
4.4.21 IdentToInt . . . 299
4.4.22 IfThen . . . 300
4.4.23 InitComponentRes . . . 300
4.4.24 InitInheritedComponent . . . 300
4.4.25 IntToIdent . . . 300
4.4.26 InvalidPoint . . . 301
4.4.27 LineStart . . . 301
4.4.28 NotifyGlobalLoading . . . 301
4.4.29 ObjectBinaryToText . . . 301
4.4.30 ObjectResourceToText . . . 302
4.4.31 ObjectTextToBinary . . . 302
4.4.32 ObjectTextToResource . . . 302
4.4.33 Point . . . 302
4.4.34 PointsEqual . . . 302
4.4.35 ReadComponentRes . . . 303
4.4.36 ReadComponentResEx . . . 303
4.4.37 ReadComponentResFile . . . 303
4.4.38 Rect . . . 304
4.4.39 RedirectFixupReferences . . . 304
4.4.40 RegisterClass . . . 304
4.4.41 RegisterClassAlias . . . 304
4.4.42 RegisterClasses . . . 305
4.4.43 RegisterComponents . . . 305
4.4.44 RegisterFindGlobalComponentProc . . . 305
4.4.45 RegisterInitComponentHandler . . . 305
4.4.46 RegisterIntegerConsts . . . 306
4.4.47 RegisterNoIcon . . . 306
4.4.48 RegisterNonActiveX . . . 306
4.4.49 RemoveFixupReferences . . . 307
4.4.50 RemoveFixups . . . 307
4.4.51 SmallPoint . . . 307
4.4.52 StartClassGroup . . . 307
4.4.53 UnRegisterClass . . . 308

4.4.54 UnRegisterClasses . . . 308
4.4.55 UnregisterFindGlobalComponentProc . . . 308
4.4.56 UnRegisterModuleClasses . . . 308
4.4.57 WriteComponentResFile . . . 309
4.5 TIdentMapEntry . . . 309
4.6 TStringItem . . . 309
4.7 EBitsError . . . 309
4.7.1 Description . . . 309
4.8 EClassNotFound . . . 309
4.8.1 Description . . . 309
4.9 EComponentError . . . 310
4.9.1 Description . . . 310
4.10 EFCreateError . . . 310
4.10.1 Description . . . 310
4.11 EFilerError . . . 310
4.11.1 Description . . . 310
4.12 EFOpenError . . . 310
4.12.1 Description . . . 310
4.13 EInvalidImage . . . 310
4.13.1 Description . . . 310
4.14 EInvalidOperation . . . 311
4.14.1 Description . . . 311
4.15 EMethodNotFound . . . 311
4.15.1 Description . . . 311
4.16 EObserver . . . 311
4.16.1 Description . . . 311
4.17 EOutOfResources . . . 311
4.17.1 Description . . . 311
4.18 EParserError . . . 311
4.18.1 Description . . . 311
4.19 EReadError . . . 311
4.19.1 Description . . . 311
4.20 EResNotFound . . . 312
4.20.1 Description . . . 312
4.21 EStreamError . . . 312
4.21.1 Description . . . 312
4.22 EStringListError . . . 312
4.22.1 Description . . . 312
4.23 EThread . . . 312
4.23.1 Description . . . 312

4.24 EThreadDestroyCalled . . . 313
4.24.1 Description . . . 313
4.25 EThreadExternalException . . . 313
4.25.1 Description . . . 313
4.26 EWriteError . . . 313
4.26.1 Description . . . 313
4.27 IDesignerNotify . . . 313
4.27.1 Description . . . 313
4.27.2 Method overview . . . 313
4.27.3 IDesignerNotify.Modified . . . 313
4.27.4 IDesignerNotify.Notification . . . 314
4.28 IFPObserved . . . 314
4.28.1 Description . . . 314
4.28.2 Method overview . . . 314
4.28.3 IFPObserved.FPOAttachObserver . . . 314
4.28.4 IFPObserved.FPODetachObserver . . . 315
4.28.5 IFPObserved.FPONotifyObservers . . . 315
4.29 IFPObserver . . . 315
4.29.1 Description . . . 315
4.29.2 Method overview . . . 315
4.29.3 IFPObserver.FPOObservedChanged . . . 316
4.30 IInterfaceComponentReference . . . 316
4.30.1 Description . . . 316
4.30.2 Method overview . . . 316
4.30.3 IInterfaceComponentReference.GetComponent . . . 316
4.31 IInterfaceList . . . 316
4.31.1 Description . . . 316
4.31.2 Method overview . . . 317
4.31.3 Property overview . . . 317
4.31.4 IInterfaceList.Get . . . 317
4.31.5 IInterfaceList.GetCapacity . . . 317
4.31.6 IInterfaceList.GetCount . . . 318
4.31.7 IInterfaceList.Put . . . 318
4.31.8 IInterfaceList.SetCapacity . . . 318
4.31.9 IInterfaceList.SetCount . . . 318
4.31.10 IInterfaceList.Clear . . . 319
4.31.11 IInterfaceList.Delete . . . 319
4.31.12 IInterfaceList.Exchange . . . 319
4.31.13 IInterfaceList.First . . . 319
4.31.14 IInterfaceList.IndexOf . . . 320

4.31.15 IInterfaceList.Add . . . 320
4.31.16 IInterfaceList.Insert . . . 320
4.31.17 IInterfaceList.Last . . . 320
4.31.18 IInterfaceList.Remove . . . 320
4.31.19 IInterfaceList.Lock . . . 321
4.31.20 IInterfaceList.Unlock . . . 321
4.31.21 IInterfaceList.Capacity . . . 321
4.31.22 IInterfaceList.Count . . . 321
4.31.23 IInterfaceList.Items . . . 322
4.32 IStreamPersist . . . 322
4.32.1 Description . . . 322
4.32.2 Method overview . . . 322
4.32.3 IStreamPersist.LoadFromStream . . . 322
4.32.4 IStreamPersist.SaveToStream . . . 322
4.33 IStringsAdapter . . . 323
4.33.1 Description . . . 323
4.33.2 Method overview . . . 323
4.33.3 IStringsAdapter.ReferenceStrings . . . 323
4.33.4 IStringsAdapter.ReleaseStrings . . . 323
4.34 IVCLComObject . . . 323
4.34.1 Description . . . 323
4.34.2 Method overview . . . 323
4.34.3 IVCLComObject.GetTypeInfoCount . . . 324
4.34.4 IVCLComObject.GetTypeInfo . . . 324
4.34.5 IVCLComObject.GetIDsOfNames . . . 324
4.34.6 IVCLComObject.Invoke . . . 324
4.34.7 IVCLComObject.SafeCallException . . . 325
4.34.8 IVCLComObject.FreeOnRelease . . . 325
4.35 TAbstractObjectReader . . . 325
4.35.1 Description . . . 325
4.35.2 Method overview . . . 326
4.35.3 TAbstractObjectReader.FlushBuffer . . . 326
4.35.4 TAbstractObjectReader.NextValue . . . 326
4.35.5 TAbstractObjectReader.ReadValue . . . 327
4.35.6 TAbstractObjectReader.BeginRootComponent . . . 327
4.35.7 TAbstractObjectReader.BeginComponent . . . 327
4.35.8 TAbstractObjectReader.BeginProperty . . . 327
4.35.9 TAbstractObjectReader.Read . . . 328
4.35.10 TAbstractObjectReader.ReadBinary . . . 328
4.35.11 TAbstractObjectReader.ReadFloat . . . 328

4.35.12 TAbstractObjectReader.ReadSingle . . . 328
4.35.13 TAbstractObjectReader.ReadDate . . . 329
4.35.14 TAbstractObjectReader.ReadCurrency . . . 329
4.35.15 TAbstractObjectReader.ReadIdent . . . 329
4.35.16 TAbstractObjectReader.ReadInt8 . . . 330
4.35.17 TAbstractObjectReader.ReadInt16 . . . 330
4.35.18 TAbstractObjectReader.ReadInt32 . . . 331
4.35.19 TAbstractObjectReader.ReadInt64 . . . 331
4.35.20 TAbstractObjectReader.ReadSet . . . 331
4.35.21 TAbstractObjectReader.ReadSignature . . . 332
4.35.22 TAbstractObjectReader.ReadStr . . . 332
4.35.23 TAbstractObjectReader.ReadString . . . 332
4.35.24 TAbstractObjectReader.ReadWideString . . . 332
4.35.25 TAbstractObjectReader.ReadUnicodeString . . . 333
4.35.26 TAbstractObjectReader.SkipComponent . . . 333
4.35.27 TAbstractObjectReader.SkipValue . . . 333
4.36 TAbstractObjectWriter . . . 333
4.36.1 Description . . . 333
4.36.2 Method overview . . . 334
4.36.3 TAbstractObjectWriter.BeginCollection . . . 334
4.36.4 TAbstractObjectWriter.BeginComponent . . . 334
4.36.5 TAbstractObjectWriter.WriteSignature . . . 334
4.36.6 TAbstractObjectWriter.BeginList . . . 335
4.36.7 TAbstractObjectWriter.EndList . . . 335
4.36.8 TAbstractObjectWriter.BeginProperty . . . 335
4.36.9 TAbstractObjectWriter.EndProperty . . . 335
4.36.10 TAbstractObjectWriter.FlushBuffer . . . 335
4.36.11 TAbstractObjectWriter.Write . . . 336
4.36.12 TAbstractObjectWriter.WriteBinary . . . 336
4.36.13 TAbstractObjectWriter.WriteBoolean . . . 336
4.36.14 TAbstractObjectWriter.WriteFloat . . . 336
4.36.15 TAbstractObjectWriter.WriteSingle . . . 336
4.36.16 TAbstractObjectWriter.WriteDate . . . 337
4.36.17 TAbstractObjectWriter.WriteCurrency . . . 337
4.36.18 TAbstractObjectWriter.WriteIdent . . . 337
4.36.19 TAbstractObjectWriter.WriteInteger . . . 337
4.36.20 TAbstractObjectWriter.WriteUInt64 . . . 337
4.36.21 TAbstractObjectWriter.WriteVariant . . . 338
4.36.22 TAbstractObjectWriter.WriteMethodName . . . 338
4.36.23 TAbstractObjectWriter.WriteSet . . . 338

4.36.24 TAbstractObjectWriter.WriteString . . . 338
4.36.25 TAbstractObjectWriter.WriteWideString . . . 338
4.36.26 TAbstractObjectWriter.WriteUnicodeString . . . 339
4.37 TBasicAction . . . 339
4.37.1 Description . . . 339
4.37.2 Method overview . . . 339
4.37.3 Property overview . . . 339
4.37.4 TBasicAction.Create . . . 339
4.37.5 TBasicAction.Destroy . . . 340
4.37.6 TBasicAction.HandlesTarget . . . 340
4.37.7 TBasicAction.UpdateTarget . . . 340
4.37.8 TBasicAction.ExecuteTarget . . . 341
4.37.9 TBasicAction.Execute . . . 341
4.37.10 TBasicAction.RegisterChanges . . . 341
4.37.11 TBasicAction.UnRegisterChanges . . . 341
4.37.12 TBasicAction.Update . . . 342
4.37.13 TBasicAction.Suspended . . . 342
4.37.14 TBasicAction.ActionComponent . . . 342
4.37.15 TBasicAction.OnExecute . . . 342
4.37.16 TBasicAction.OnUpdate . . . 343
4.38 TBasicActionLink . . . 343
4.38.1 Description . . . 343
4.38.2 Method overview . . . 343
4.38.3 Property overview . . . 343
4.38.4 TBasicActionLink.Create . . . 343
4.38.5 TBasicActionLink.Destroy . . . 344
4.38.6 TBasicActionLink.Execute . . . 344
4.38.7 TBasicActionLink.Update . . . 344
4.38.8 TBasicActionLink.Action . . . 345
4.38.9 TBasicActionLink.OnChange . . . 345
4.39 TBinaryObjectReader . . . 345
4.39.1 Description . . . 345
4.39.2 Method overview . . . 346
4.39.3 TBinaryObjectReader.Create . . . 346
4.39.4 TBinaryObjectReader.Destroy . . . 346
4.39.5 TBinaryObjectReader.NextValue . . . 347
4.39.6 TBinaryObjectReader.ReadValue . . . 347
4.39.7 TBinaryObjectReader.BeginRootComponent . . . 347
4.39.8 TBinaryObjectReader.BeginComponent . . . 347
4.39.9 TBinaryObjectReader.BeginProperty . . . 348

4.39.10 TBinaryObjectReader.Read . . . 348
4.39.11 TBinaryObjectReader.ReadBinary . . . 348
4.39.12 TBinaryObjectReader.ReadFloat . . . 348
4.39.13 TBinaryObjectReader.ReadSingle . . . 348
4.39.14 TBinaryObjectReader.ReadDate . . . 349
4.39.15 TBinaryObjectReader.ReadCurrency . . . 349
4.39.16 TBinaryObjectReader.ReadIdent . . . 349
4.39.17 TBinaryObjectReader.ReadInt8 . . . 349
4.39.18 TBinaryObjectReader.ReadInt16 . . . 349
4.39.19 TBinaryObjectReader.ReadInt32 . . . 350
4.39.20 TBinaryObjectReader.ReadInt64 . . . 350
4.39.21 TBinaryObjectReader.ReadSet . . . 350
4.39.22 TBinaryObjectReader.ReadSignature . . . 350
4.39.23 TBinaryObjectReader.ReadStr . . . 351
4.39.24 TBinaryObjectReader.ReadString . . . 351
4.39.25 TBinaryObjectReader.ReadWideString . . . 351
4.39.26 TBinaryObjectReader.ReadUnicodeString . . . 351
4.39.27 TBinaryObjectReader.SkipComponent . . . 352
4.39.28 TBinaryObjectReader.SkipValue . . . 352
4.40 TBinaryObjectWriter . . . 352
4.40.1 Description . . . 352
4.40.2 Method overview . . . 353
4.40.3 TBinaryObjectWriter.Create . . . 353
4.40.4 TBinaryObjectWriter.Destroy . . . 353
4.40.5 TBinaryObjectWriter.WriteSignature . . . 353
4.40.6 TBinaryObjectWriter.FlushBuffer . . . 354
4.40.7 TBinaryObjectWriter.BeginCollection . . . 354
4.40.8 TBinaryObjectWriter.BeginComponent . . . 354
4.40.9 TBinaryObjectWriter.BeginList . . . 354
4.40.10 TBinaryObjectWriter.EndList . . . 354
4.40.11 TBinaryObjectWriter.BeginProperty . . . 355
4.40.12 TBinaryObjectWriter.EndProperty . . . 355
4.40.13 TBinaryObjectWriter.Write . . . 355
4.40.14 TBinaryObjectWriter.WriteBinary . . . 355
4.40.15 TBinaryObjectWriter.WriteBoolean . . . 355
4.40.16 TBinaryObjectWriter.WriteFloat . . . 355
4.40.17 TBinaryObjectWriter.WriteSingle . . . 356
4.40.18 TBinaryObjectWriter.WriteDate . . . 356
4.40.19 TBinaryObjectWriter.WriteCurrency . . . 356
4.40.20 TBinaryObjectWriter.WriteIdent . . . 356

4.40.21 TBinaryObjectWriter.WriteInteger . . . 356
4.40.22 TBinaryObjectWriter.WriteUInt64 . . . 356
4.40.23 TBinaryObjectWriter.WriteMethodName . . . 357
4.40.24 TBinaryObjectWriter.WriteSet . . . 357
4.40.25 TBinaryObjectWriter.WriteStr . . . 357
4.40.26 TBinaryObjectWriter.WriteString . . . 357
4.40.27 TBinaryObjectWriter.WriteWideString . . . 357
4.40.28 TBinaryObjectWriter.WriteUnicodeString . . . 357
4.40.29 TBinaryObjectWriter.WriteVariant . . . 358
4.41 TBits . . . 358
4.41.1 Description . . . 358
4.41.2 Method overview . . . 358
4.41.3 Property overview . . . 358
4.41.4 TBits.Create . . . 359
4.41.5 TBits.Destroy . . . 359
4.41.6 TBits.GetFSize . . . 359
4.41.7 TBits.SetOn . . . 359
4.41.8 TBits.Clear . . . 360
4.41.9 TBits.Clearall . . . 360
4.41.10 TBits.CopyBits . . . 360
4.41.11 TBits.AndBits . . . 360
4.41.12 TBits.OrBits . . . 361
4.41.13 TBits.XorBits . . . 361
4.41.14 TBits.NotBits . . . 361
4.41.15 TBits.Get . . . 362
4.41.16 TBits.Grow . . . 362
4.41.17 TBits.Equals . . . 362
4.41.18 TBits.SetIndex . . . 362
4.41.19 TBits.FindFirstBit . . . 363
4.41.20 TBits.FindNextBit . . . 363
4.41.21 TBits.FindPrevBit . . . 363
4.41.22 TBits.OpenBit . . . 364
4.41.23 TBits.Bits . . . 364
4.41.24 TBits.Size . . . 364
4.42 TBytesStream . . . 364
4.42.1 Description . . . 364
4.42.2 Method overview . . . 365
4.42.3 Property overview . . . 365
4.42.4 TBytesStream.Create . . . 365
4.42.5 TBytesStream.Bytes . . . 365

4.43 TCollection . . . 365
4.43.1 Description . . . 365
4.43.2 Method overview . . . 366
4.43.3 Property overview . . . 366
4.43.4 TCollection.Create . . . 366
4.43.5 TCollection.Destroy . . . 366
4.43.6 TCollection.Owner . . . 367
4.43.7 TCollection.Add . . . 367
4.43.8 TCollection.Assign . . . 367
4.43.9 TCollection.BeginUpdate . . . 367
4.43.10 TCollection.Clear . . . 368
4.43.11 TCollection.EndUpdate . . . 368
4.43.12 TCollection.Delete . . . 368
4.43.13 TCollection.GetEnumerator . . . 369
4.43.14 TCollection.GetNamePath . . . 369
4.43.15 TCollection.Insert . . . 369
4.43.16 TCollection.FindItemID . . . 369
4.43.17 TCollection.Exchange . . . 370
4.43.18 TCollection.Move . . . 370
4.43.19 TCollection.Sort . . . 370
4.43.20 TCollection.Count . . . 371
4.43.21 TCollection.ItemClass . . . 371
4.43.22 TCollection.Items . . . 371
4.44 TCollectionEnumerator . . . 372
4.44.1 Description . . . 372
4.44.2 Method overview . . . 372
4.44.3 Property overview . . . 372
4.44.4 TCollectionEnumerator.Create . . . 372
4.44.5 TCollectionEnumerator.GetCurrent . . . 372
4.44.6 TCollectionEnumerator.MoveNext . . . 372
4.44.7 TCollectionEnumerator.Current . . . 373
4.45 TCollectionItem . . . 373
4.45.1 Description . . . 373
4.45.2 Method overview . . . 373
4.45.3 Property overview . . . 373
4.45.4 TCollectionItem.Create . . . 374
4.45.5 TCollectionItem.Destroy . . . 374
4.45.6 TCollectionItem.GetNamePath . . . 374
4.45.7 TCollectionItem.Collection . . . 374
4.45.8 TCollectionItem.ID . . . 375

4.45.9 TCollectionItem.Index . . . 375
4.45.10 TCollectionItem.DisplayName . . . 375
4.46 TComponent . . . 376
4.46.1 Description . . . 376
4.46.2 Interfaces overview . . . 376
4.46.3 Method overview . . . 376
4.46.4 Property overview . . . 377
4.46.5 TComponent.Notification . . . 377
4.46.6 TComponent.WriteState . . . 377
4.46.7 TComponent.Create . . . 378
4.46.8 TComponent.Destroy . . . 378
4.46.9 TComponent.BeforeDestruction . . . 378
4.46.10 TComponent.DestroyComponents . . . 378
4.46.11 TComponent.Destroying . . . 379
4.46.12 TComponent.ExecuteAction . . . 379
4.46.13 TComponent.FindComponent . . . 379
4.46.14 TComponent.FreeNotification . . . 379
4.46.15 TComponent.RemoveFreeNotification . . . 380
4.46.16 TComponent.FreeOnRelease . . . 380
4.46.17 TComponent.GetEnumerator . . . 380
4.46.18 TComponent.GetNamePath . . . 380
4.46.19 TComponent.GetParentComponent . . . 380
4.46.20 TComponent.HasParent . . . 381
4.46.21 TComponent.InsertComponent . . . 381
4.46.22 TComponent.RemoveComponent . . . 381
4.46.23 TComponent.SafeCallException . . . 381
4.46.24 TComponent.SetSubComponent . . . 382
4.46.25 TComponent.UpdateAction . . . 382
4.46.26 TComponent.IsImplementorOf . . . 382
4.46.27 TComponent.ReferenceInterface . . . 382
4.46.28 TComponent.ComObject . . . 383
4.46.29 TComponent.Components . . . 383
4.46.30 TComponent.ComponentCount . . . 383
4.46.31 TComponent.ComponentIndex . . . 383
4.46.32 TComponent.ComponentState . . . 384
4.46.33 TComponent.ComponentStyle . . . 384
4.46.34 TComponent.DesignInfo . . . 384
4.46.35 TComponent.Owner . . . 385
4.46.36 TComponent.VCLComObject . . . 385
4.46.37 TComponent.Name . . . 385

4.46.38 TComponent.Tag . . . 385
4.47 TComponentEnumerator . . . 386
4.47.1 Description . . . 386
4.47.2 Method overview . . . 386
4.47.3 Property overview . . . 386
4.47.4 TComponentEnumerator.Create . . . 386
4.47.5 TComponentEnumerator.GetCurrent . . . 386
4.47.6 TComponentEnumerator.MoveNext . . . 387
4.47.7 TComponentEnumerator.Current . . . 387
4.48 TCustomMemoryStream . . . 387
4.48.1 Description . . . 387
4.48.2 Method overview . . . 387
4.48.3 Property overview . . . 388
4.48.4 TCustomMemoryStream.Read . . . 388
4.48.5 TCustomMemoryStream.Seek . . . 388
4.48.6 TCustomMemoryStream.SaveToStream . . . 388
4.48.7 TCustomMemoryStream.SaveToFile . . . 389
4.48.8 TCustomMemoryStream.Memory . . . 389
4.49 TDataModule . . . 389
4.49.1 Description . . . 389
4.49.2 Method overview . . . 390
4.49.3 Property overview . . . 390
4.49.4 TDataModule.Create . . . 390
4.49.5 TDataModule.CreateNew . . . 390
4.49.6 TDataModule.Destroy . . . 391
4.49.7 TDataModule.AfterConstruction . . . 391
4.49.8 TDataModule.BeforeDestruction . . . 391
4.49.9 TDataModule.DesignOffset . . . 391
4.49.10 TDataModule.DesignSize . . . 392
4.49.11 TDataModule.DesignPPI . . . 392
4.49.12 TDataModule.OnCreate . . . 392
4.49.13 TDataModule.OnDestroy . . . 392
4.49.14 TDataModule.OldCreateOrder . . . 393
4.50 TFiler . . . 393
4.50.1 Description . . . 393
4.50.2 Method overview . . . 393
4.50.3 Property overview . . . 393
4.50.4 TFiler.DefineProperty . . . 393
4.50.5 TFiler.DefineBinaryProperty . . . 394
4.50.6 TFiler.FlushBuffer . . . 394

4.50.7 TFiler.Root . . . 394
4.50.8 TFiler.LookupRoot . . . 394
4.50.9 TFiler.Ancestor . . . 395
4.50.10 TFiler.IgnoreChildren . . . 395
4.51 TFileStream . . . 395
4.51.1 Description . . . 395
4.51.2 Method overview . . . 395
4.51.3 Property overview . . . 395
4.51.4 TFileStream.Create . . . 396
4.51.5 TFileStream.Destroy . . . 396
4.51.6 TFileStream.FileName . . . 397
4.52 TFPList . . . 397
4.52.1 Description . . . 397
4.52.2 Method overview . . . 397
4.52.3 Property overview . . . 398
4.52.4 TFPList.Destroy . . . 398
4.52.5 TFPList.AddList . . . 398
4.52.6 TFPList.Add . . . 398
4.52.7 TFPList.Clear . . . 398
4.52.8 TFPList.Delete . . . 399
4.52.9 TFPList.Error . . . 399
4.52.10 TFPList.Exchange . . . 399
4.52.11 TFPList.Expand . . . 399
4.52.12 TFPList.Extract . . . 400
4.52.13 TFPList.First . . . 400
4.52.14 TFPList.GetEnumerator . . . 400
4.52.15 TFPList.IndexOf . . . 400
4.52.16 TFPList.IndexOfItem . . . 401
4.52.17 TFPList.Insert . . . 401
4.52.18 TFPList.Last . . . 401
4.52.19 TFPList.Move . . . 401
4.52.20 TFPList.Assign . . . 402
4.52.21 TFPList.Remove . . . 402
4.52.22 TFPList.Pack . . . 402
4.52.23 TFPList.Sort . . . 402
4.52.24 TFPList.ForEachCall . . . 403
4.52.25 TFPList.Capacity . . . 403
4.52.26 TFPList.Count . . . 403
4.52.27 TFPList.Items . . . 404
4.52.28 TFPList.List . . . 404

4.53 TFPListEnumerator . . . 404
4.53.1 Description . . . 404
4.53.2 Method overview . . . 404
4.53.3 Property overview . . . 404
4.53.4 TFPListEnumerator.Create . . . 404
4.53.5 TFPListEnumerator.GetCurrent . . . 405
4.53.6 TFPListEnumerator.MoveNext . . . 405
4.53.7 TFPListEnumerator.Current . . . 405
4.54 THandleStream . . . 406
4.54.1 Description . . . 406
4.54.2 Method overview . . . 406
4.54.3 Property overview . . . 406
4.54.4 THandleStream.Create . . . 406
4.54.5 THandleStream.Read . . . 406
4.54.6 THandleStream.Write . . . 407
4.54.7 THandleStream.Seek . . . 407
4.54.8 THandleStream.Handle . . . 407
4.55 TInterfacedPersistent . . . 408
4.55.1 Description . . . 408
4.55.2 Interfaces overview . . . 408
4.55.3 Method overview . . . 408
4.55.4 TInterfacedPersistent.QueryInterface . . . 408
4.55.5 TInterfacedPersistent.AfterConstruction . . . 408
4.56 TInterfaceList . . . 408
4.56.1 Description . . . 408
4.56.2 Interfaces overview . . . 409
4.56.3 Method overview . . . 409
4.56.4 Property overview . . . 409
4.56.5 TInterfaceList.Create . . . 409
4.56.6 TInterfaceList.Destroy . . . 409
4.56.7 TInterfaceList.Clear . . . 410
4.56.8 TInterfaceList.Delete . . . 410
4.56.9 TInterfaceList.Exchange . . . 410
4.56.10 TInterfaceList.First . . . 410
4.56.11 TInterfaceList.GetEnumerator . . . 411
4.56.12 TInterfaceList.IndexOf . . . 411
4.56.13 TInterfaceList.Add . . . 411
4.56.14 TInterfaceList.Insert . . . 411
4.56.15 TInterfaceList.Last . . . 412
4.56.16 TInterfaceList.Remove . . . 412

4.56.17 TInterfaceList.Lock . . . 412
4.56.18 TInterfaceList.Unlock . . . 412
4.56.19 TInterfaceList.Expand . . . 413
4.56.20 TInterfaceList.Capacity . . . 413
4.56.21 TInterfaceList.Count . . . 413
4.56.22 TInterfaceList.Items . . . 413
4.57 TInterfaceListEnumerator . . . 414
4.57.1 Description . . . 414
4.57.2 Method overview . . . 414
4.57.3 Property overview . . . 414
4.57.4 TInterfaceListEnumerator.Create . . . 414
4.57.5 TInterfaceListEnumerator.GetCurrent . . . 414
4.57.6 TInterfaceListEnumerator.MoveNext . . . 415
4.57.7 TInterfaceListEnumerator.Current . . . 415
4.58 TList . . . 415
4.58.1 Description . . . 415
4.58.2 Interfaces overview . . . 415
4.58.3 Method overview . . . 416
4.58.4 Property overview . . . 416
4.58.5 TList.Create . . . 416
4.58.6 TList.Destroy . . . 416
4.58.7 TList.FPOAttachObserver . . . 417
4.58.8 TList.FPODetachObserver . . . 417
4.58.9 TList.FPONotifyObservers . . . 417
4.58.10 TList.AddList . . . 418
4.58.11 TList.Add . . . 418
4.58.12 TList.Clear . . . 418
4.58.13 TList.Delete . . . 418
4.58.14 TList.Error . . . 419
4.58.15 TList.Exchange . . . 419
4.58.16 TList.Expand . . . 419
4.58.17 TList.Extract . . . 419
4.58.18 TList.First . . . 420
4.58.19 TList.GetEnumerator . . . 420
4.58.20 TList.IndexOf . . . 420
4.58.21 TList.Insert . . . 420
4.58.22 TList.Last . . . 421
4.58.23 TList.Move . . . 421
4.58.24 TList.Assign . . . 421
4.58.25 TList.Remove . . . 421

4.58.26 TList.Pack . . . 422
4.58.27 TList.Sort . . . 422
4.58.28 TList.Capacity . . . 422
4.58.29 TList.Count . . . 423
4.58.30 TList.Items . . . 423
4.58.31 TList.List . . . 423
4.59 TListEnumerator . . . 423
4.59.1 Description . . . 423
4.59.2 Method overview . . . 423
4.59.3 Property overview . . . 424
4.59.4 TListEnumerator.Create . . . 424
4.59.5 TListEnumerator.GetCurrent . . . 424
4.59.6 TListEnumerator.MoveNext . . . 424
4.59.7 TListEnumerator.Current . . . 424
4.60 TMemoryStream . . . 425
4.60.1 Description . . . 425
4.60.2 Method overview . . . 425
4.60.3 TMemoryStream.Destroy . . . 425
4.60.4 TMemoryStream.Clear . . . 425
4.60.5 TMemoryStream.LoadFromStream . . . 425
4.60.6 TMemoryStream.LoadFromFile . . . 426
4.60.7 TMemoryStream.SetSize . . . 426
4.60.8 TMemoryStream.Write . . . 426
4.61 TOwnedCollection . . . 427
4.61.1 Description . . . 427
4.61.2 Method overview . . . 427
4.61.3 TOwnedCollection.Create . . . 427
4.62 TOwnerStream . . . 427
4.62.1 Description . . . 427
4.62.2 Method overview . . . 428
4.62.3 Property overview . . . 428
4.62.4 TOwnerStream.Create . . . 428
4.62.5 TOwnerStream.Destroy . . . 428
4.62.6 TOwnerStream.Source . . . 428
4.62.7 TOwnerStream.SourceOwner . . . 429
4.63 TParser . . . 429
4.63.1 Description . . . 429
4.63.2 Method overview . . . 429
4.63.3 Property overview . . . 430
4.63.4 TParser.Create . . . 430

4.63.5 TParser.Destroy . . . 430
4.63.6 TParser.CheckToken . . . 430
4.63.7 TParser.CheckTokenSymbol . . . 430
4.63.8 TParser.Error . . . 431
4.63.9 TParser.ErrorFmt . . . 431
4.63.10 TParser.ErrorStr . . . 431
4.63.11 TParser.HexToBinary . . . 431
4.63.12 TParser.NextToken . . . 432
4.63.13 TParser.SourcePos . . . 432
4.63.14 TParser.TokenComponentIdent . . . 432
4.63.15 TParser.TokenFloat . . . 433
4.63.16 TParser.TokenInt . . . 433
4.63.17 TParser.TokenString . . . 433
4.63.18 TParser.TokenWideString . . . 434
4.63.19 TParser.TokenSymbolIs . . . 434
4.63.20 TParser.FloatType . . . 434
4.63.21 TParser.SourceLine . . . 435
4.63.22 TParser.Token . . . 435
4.64 TPersistent . . . 435
4.64.1 Description . . . 435
4.64.2 Interfaces overview . . . 436
4.64.3 Method overview . . . 436
4.64.4 TPersistent.Destroy . . . 436
4.64.5 TPersistent.Assign . . . 436
4.64.6 TPersistent.FPOAttachObserver . . . 437
4.64.7 TPersistent.FPODetachObserver . . . 437
4.64.8 TPersistent.FPONotifyObservers . . . 437
4.64.9 TPersistent.GetNamePath . . . 437
4.65 TProxyStream . . . 438
4.65.1 Description . . . 438
4.65.2 Method overview . . . 438
4.65.3 TProxyStream.Create . . . 438
4.65.4 TProxyStream.Read . . . 438
4.65.5 TProxyStream.Write . . . 438
4.65.6 TProxyStream.Seek . . . 438
4.65.7 TProxyStream.Check . . . 439
4.66 TRawByteStringStream . . . 439
4.66.1 Description . . . 439
4.66.2 Method overview . . . 439
4.66.3 TRawByteStringStream.Create . . . 439

4.66.4 TRawByteStringStream.DataString . . . 439
4.66.5 TRawByteStringStream.ReadString . . . 440
4.66.6 TRawByteStringStream.WriteString . . . 440
4.67 TReader . . . 440
4.67.1 Description . . . 440
4.67.2 Method overview . . . 441
4.67.3 Property overview . . . 442
4.67.4 TReader.Create . . . 442
4.67.5 TReader.Destroy . . . 442
4.67.6 TReader.FlushBuffer . . . 442
4.67.7 TReader.BeginReferences . . . 443
4.67.8 TReader.CheckValue . . . 443
4.67.9 TReader.DefineProperty . . . 443
4.67.10 TReader.DefineBinaryProperty . . . 443
4.67.11 TReader.EndOfList . . . 443
4.67.12 TReader.EndReferences . . . 444
4.67.13 TReader.FixupReferences . . . 444
4.67.14 TReader.NextValue . . . 444
4.67.15 TReader.Read . . . 444
4.67.16 TReader.ReadBoolean . . . 444
4.67.17 TReader.ReadChar . . . 445
4.67.18 TReader.ReadWideChar . . . 445
4.67.19 TReader.ReadUnicodeChar . . . 445
4.67.20 TReader.ReadCollection . . . 445
4.67.21 TReader.ReadComponent . . . 445
4.67.22 TReader.ReadComponents . . . 446
4.67.23 TReader.ReadFloat . . . 446
4.67.24 TReader.ReadSingle . . . 446
4.67.25 TReader.ReadDate . . . 446
4.67.26 TReader.ReadCurrency . . . 446
4.67.27 TReader.ReadIdent . . . 447
4.67.28 TReader.ReadInteger . . . 447
4.67.29 TReader.ReadInt64 . . . 447
4.67.30 TReader.ReadSet . . . 447
4.67.31 TReader.ReadListBegin . . . 447
4.67.32 TReader.ReadListEnd . . . 448
4.67.33 TReader.ReadRootComponent . . . 448
4.67.34 TReader.ReadVariant . . . 448
4.67.35 TReader.ReadSignature . . . 448
4.67.36 TReader.ReadString . . . 448

4.67.37 TReader.ReadWideString . . . 449
4.67.38 TReader.ReadUnicodeString . . . 449
4.67.39 TReader.ReadValue . . . 449
4.67.40 TReader.CopyValue . . . 449
4.67.41 TReader.Driver . . . 449
4.67.42 TReader.Owner . . . 450
4.67.43 TReader.Parent . . . 450
4.67.44 TReader.OnError . . . 450
4.67.45 TReader.OnPropertyNotFound . . . 450
4.67.46 TReader.OnFindMethod . . . 451
4.67.47 TReader.OnSetMethodProperty . . . 451
4.67.48 TReader.OnSetName . . . 451
4.67.49 TReader.OnReferenceName . . . 451
4.67.50 TReader.OnAncestorNotFound . . . 451
4.67.51 TReader.OnCreateComponent . . . 452
4.67.52 TReader.OnFindComponentClass . . . 452
4.67.53 TReader.OnReadStringProperty . . . 452
4.68 TRecall . . . 452
4.68.1 Description . . . 452
4.68.2 Method overview . . . 453
4.68.3 Property overview . . . 453
4.68.4 TRecall.Create . . . 453
4.68.5 TRecall.Destroy . . . 453
4.68.6 TRecall.Store . . . 453
4.68.7 TRecall.Forget . . . 454
4.68.8 TRecall.Reference . . . 454
4.69 TResourceStream . . . 454
4.69.1 Description . . . 454
4.69.2 Method overview . . . 454
4.69.3 TResourceStream.Create . . . 455
4.69.4 TResourceStream.CreateFromID . . . 455
4.69.5 TResourceStream.Destroy . . . 455
4.70 TStream . . . 455
4.70.1 Description . . . 455
4.70.2 Method overview . . . 456
4.70.3 Property overview . . . 456
4.70.4 TStream.Read . . . 456
4.70.5 TStream.Write . . . 457
4.70.6 TStream.Seek . . . 457
4.70.7 TStream.ReadBuffer . . . 458

4.70.8 TStream.WriteBuffer . . . 458
4.70.9 TStream.CopyFrom . . . 459
4.70.10 TStream.ReadComponent . . . 459
4.70.11 TStream.ReadComponentRes . . . 459
4.70.12 TStream.WriteComponent . . . 460
4.70.13 TStream.WriteComponentRes . . . 460
4.70.14 TStream.WriteDescendent . . . 460
4.70.15 TStream.WriteDescendentRes . . . 460
4.70.16 TStream.WriteResourceHeader . . . 461
4.70.17 TStream.FixupResourceHeader . . . 461
4.70.18 TStream.ReadResHeader . . . 461
4.70.19 TStream.ReadByte . . . 461
4.70.20 TStream.ReadWord . . . 462
4.70.21 TStream.ReadDWord . . . 462
4.70.22 TStream.ReadQWord . . . 462
4.70.23 TStream.ReadAnsiString . . . 463
4.70.24 TStream.WriteByte . . . 463
4.70.25 TStream.WriteWord . . . 463
4.70.26 TStream.WriteDWord . . . 463
4.70.27 TStream.WriteQWord . . . 464
4.70.28 TStream.WriteAnsiString . . . 464
4.70.29 TStream.Position . . . 464
4.70.30 TStream.Size . . . 465
4.71 TStreamAdapter . . . 465
4.71.1 Description . . . 465
4.71.2 Interfaces overview . . . 465
4.71.3 Method overview . . . 465
4.71.4 Property overview . . . 466
4.71.5 TStreamAdapter.Create . . . 466
4.71.6 TStreamAdapter.Destroy . . . 466
4.71.7 TStreamAdapter.Read . . . 466
4.71.8 TStreamAdapter.Write . . . 466
4.71.9 TStreamAdapter.Seek . . . 467
4.71.10 TStreamAdapter.SetSize . . . 467
4.71.11 TStreamAdapter.CopyTo . . . 467
4.71.12 TStreamAdapter.Commit . . . 468
4.71.13 TStreamAdapter.Revert . . . 468
4.71.14 TStreamAdapter.LockRegion . . . 468
4.71.15 TStreamAdapter.UnlockRegion . . . 468
4.71.16 TStreamAdapter.Stat . . . 469

4.71.17 TStreamAdapter.Clone . . . 469
4.71.18 TStreamAdapter.Stream . . . 469
4.71.19 TStreamAdapter.StreamOwnership . . . 469
4.72 TStringList . . . 470
4.72.1 Description . . . 470
4.72.2 Method overview . . . 470
4.72.3 Property overview . . . 470
4.72.4 TStringList.Create . . . 470
4.72.5 TStringList.Destroy . . . 470
4.72.6 TStringList.Add . . . 471
4.72.7 TStringList.Clear . . . 471
4.72.8 TStringList.Delete . . . 471
4.72.9 TStringList.Exchange . . . 471
4.72.10 TStringList.Find . . . 472
4.72.11 TStringList.IndexOf . . . 472
4.72.12 TStringList.Insert . . . 472
4.72.13 TStringList.Sort . . . 473
4.72.14 TStringList.CustomSort . . . 473
4.72.15 TStringList.Duplicates . . . 473
4.72.16 TStringList.Sorted . . . 474
4.72.17 TStringList.CaseSensitive . . . 474
4.72.18 TStringList.OnChange . . . 474
4.72.19 TStringList.OnChanging . . . 475
4.72.20 TStringList.OwnsObjects . . . 475
4.72.21 TStringList.SortStyle . . . 475
4.73 TStrings . . . 475
4.73.1 Description . . . 475
4.73.2 Method overview . . . 477
4.73.3 Property overview . . . 478
4.73.4 TStrings.Create . . . 478
4.73.5 TStrings.Destroy . . . 479
4.73.6 TStrings.ToObjectArray . . . 479
4.73.7 TStrings.ToStringArray . . . 479
4.73.8 TStrings.Add . . . 480
4.73.9 TStrings.AddObject . . . 480
4.73.10 TStrings.AddPair . . . 480
4.73.11 TStrings.AddStrings . . . 481
4.73.12 TStrings.SetStrings . . . 481
4.73.13 TStrings.AddText . . . 481
4.73.14 TStrings.AddCommaText . . . 481

4.73.15 TStrings.AddDelimitedtext . . . 482
4.73.16 TStrings.Append . . . 482
4.73.17 TStrings.Assign . . . 482
4.73.18 TStrings.BeginUpdate . . . 483
4.73.19 TStrings.Clear . . . 483
4.73.20 TStrings.Delete . . . 483
4.73.21 TStrings.EndUpdate . . . 484
4.73.22 TStrings.Equals . . . 484
4.73.23 TStrings.Exchange . . . 484
4.73.24 TStrings.ExtractName . . . 485
4.73.25 TStrings.Filter . . . 485
4.73.26 TStrings.Fill . . . 485
4.73.27 TStrings.ForEach . . . 485
4.73.28 TStrings.GetEnumerator . . . 486
4.73.29 TStrings.GetNameValue . . . 486
4.73.30 TStrings.GetText . . . 486
4.73.31 TStrings.IndexOf . . . 486
4.73.32 TStrings.IndexOfName . . . 487
4.73.33 TStrings.IndexOfObject . . . 487
4.73.34 TStrings.Insert . . . 487
4.73.35 TStrings.InsertObject . . . 488
4.73.36 TStrings.LastIndexOf . . . 488
4.73.37 TStrings.LoadFromFile . . . 488
4.73.38 TStrings.LoadFromStream . . . 489
4.73.39 TStrings.Map . . . 489
4.73.40 TStrings.Move . . . 489
4.73.41 TStrings.Pop . . . 490
4.73.42 TStrings.Reduce . . . 490
4.73.43 TStrings.Reverse . . . 490
4.73.44 TStrings.SaveToFile . . . 491
4.73.45 TStrings.SaveToStream . . . 491
4.73.46 TStrings.Shift . . . 491
4.73.47 TStrings.Slice . . . 492
4.73.48 TStrings.SetText . . . 492
4.73.49 TStrings.AlwaysQuote . . . 492
4.73.50 TStrings.Capacity . . . 492
4.73.51 TStrings.CommaText . . . 493
4.73.52 TStrings.Count . . . 493
4.73.53 TStrings.DefaultEncoding . . . 494
4.73.54 TStrings.DelimitedText . . . 494

4.73.55 TStrings.Delimiter . . . 495
4.73.56 TStrings.Encoding . . . 495
4.73.57 TStrings.LineBreak . . . 495
4.73.58 TStrings.MissingNameValueSeparatorAction . . . 495
4.73.59 TStrings.Names . . . 496
4.73.60 TStrings.NameValueSeparator . . . 496
4.73.61 TStrings.Objects . . . 496
4.73.62 TStrings.Options . . . 497
4.73.63 TStrings.QuoteChar . . . 497
4.73.64 TStrings.SkipLastLineBreak . . . 497
4.73.65 TStrings.TrailingLineBreak . . . 497
4.73.66 TStrings.StrictDelimiter . . . 498
4.73.67 TStrings.Strings . . . 498
4.73.68 TStrings.StringsAdapter . . . 498
4.73.69 TStrings.Text . . . 499
4.73.70 TStrings.TextLineBreakStyle . . . 499
4.73.71 TStrings.UseLocale . . . 499
4.73.72 TStrings.ValueFromIndex . . . 500
4.73.73 TStrings.Values . . . 500
4.73.74 TStrings.WriteBOM . . . 500
4.74 TStringsEnumerator . . . 501
4.74.1 Description . . . 501
4.74.2 Method overview . . . 501
4.74.3 Property overview . . . 501
4.74.4 TStringsEnumerator.Create . . . 501
4.74.5 TStringsEnumerator.GetCurrent . . . 501
4.74.6 TStringsEnumerator.MoveNext . . . 501
4.74.7 TStringsEnumerator.Current . . . 502
4.75 TStringStream . . . 502
4.75.1 Description . . . 502
4.75.2 Method overview . . . 502
4.75.3 Property overview . . . 503
4.75.4 TStringStream.Create . . . 503
4.75.5 TStringStream.CreateRaw . . . 503
4.75.6 TStringStream.Destroy . . . 503
4.75.7 TStringStream.ReadUnicodeString . . . 504
4.75.8 TStringStream.WriteUnicodeString . . . 504
4.75.9 TStringStream.ReadAnsiString . . . 504
4.75.10 TStringStream.WriteAnsiString . . . 504
4.75.11 TStringStream.ReadString . . . 505

4.75.12 TStringStream.WriteString . . . 505
4.75.13 TStringStream.DataString . . . 505
4.75.14 TStringStream.UnicodeDataString . . . 505
4.75.15 TStringStream.OwnsEncoding . . . 506
4.75.16 TStringStream.Encoding . . . 506
4.76 TTextObjectWriter . . . 506
4.76.1 Description . . . 506
4.77 TThread . . . 506
4.77.1 Description . . . 506
4.77.2 Method overview . . . 507
4.77.3 Property overview . . . 507
4.77.4 TThread.Execute . . . 507
4.77.5 TThread.Synchronize . . . 508
4.77.6 TThread.Queue . . . 508
4.77.7 TThread.ForceQueue . . . 508
4.77.8 TThread.Create . . . 509
4.77.9 TThread.Destroy . . . 509
4.77.10 TThread.CreateAnonymousThread . . . 509
4.77.11 TThread.NameThreadForDebugging . . . 509
4.77.12 TThread.SetReturnValue . . . 510
4.77.13 TThread.CheckTerminated . . . 510
4.77.14 TThread.RemoveQueuedEvents . . . 510
4.77.15 TThread.SpinWait . . . 511
4.77.16 TThread.Sleep . . . 511
4.77.17 TThread.Yield . . . 511
4.77.18 TThread.GetSystemTimes . . . 511
4.77.19 TThread.GetTickCount . . . 512
4.77.20 TThread.GetTickCount64 . . . 512
4.77.21 TThread.ExecuteInThread . . . 512
4.77.22 TThread.AfterConstruction . . . 517
4.77.23 TThread.Start . . . 517
4.77.24 TThread.Resume . . . 517
4.77.25 TThread.Suspend . . . 517
4.77.26 TThread.Terminate . . . 518
4.77.27 TThread.WaitFor . . . 518
4.77.28 TThread.CurrentThread . . . 518
4.77.29 TThread.ProcessorCount . . . 519
4.77.30 TThread.IsSingleProcessor . . . 519
4.77.31 TThread.FreeOnTerminate . . . 519
4.77.32 TThread.Handle . . . 520

4.77.33 TThread.ExternalThread . . . 520
4.77.34 TThread.Priority . . . 520
4.77.35 TThread.Suspended . . . 520
4.77.36 TThread.Finished . . . 521
4.77.37 TThread.ThreadID . . . 521
4.77.38 TThread.OnTerminate . . . 521
4.77.39 TThread.FatalException . . . 521
4.78 TThreadList . . . 522
4.78.1 Description . . . 522
4.78.2 Method overview . . . 522
4.78.3 Property overview . . . 522
4.78.4 TThreadList.Create . . . 522
4.78.5 TThreadList.Destroy . . . 522
4.78.6 TThreadList.Add . . . 523
4.78.7 TThreadList.Clear . . . 523
4.78.8 TThreadList.LockList . . . 523
4.78.9 TThreadList.Remove . . . 523
4.78.10 TThreadList.UnlockList . . . 524
4.78.11 TThreadList.Duplicates . . . 524
4.79 TWriter . . . 524
4.79.1 Description . . . 524
4.79.2 Method overview . . . 525
4.79.3 Property overview . . . 525
4.79.4 TWriter.Create . . . 525
4.79.5 TWriter.Destroy . . . 526
4.79.6 TWriter.FlushBuffer . . . 526
4.79.7 TWriter.DefineProperty . . . 526
4.79.8 TWriter.DefineBinaryProperty . . . 526
4.79.9 TWriter.Write . . . 526
4.79.10 TWriter.WriteBoolean . . . 527
4.79.11 TWriter.WriteCollection . . . 527
4.79.12 TWriter.WriteComponent . . . 527
4.79.13 TWriter.WriteChar . . . 527
4.79.14 TWriter.WriteWideChar . . . 527
4.79.15 TWriter.WriteDescendent . . . 528
4.79.16 TWriter.WriteFloat . . . 528
4.79.17 TWriter.WriteSingle . . . 528
4.79.18 TWriter.WriteDate . . . 528
4.79.19 TWriter.WriteCurrency . . . 528
4.79.20 TWriter.WriteIdent . . . 529

4.79.21 TWriter.WriteInteger . . . 529
4.79.22 TWriter.WriteSet . . . 529
4.79.23 TWriter.WriteListBegin . . . 529
4.79.24 TWriter.WriteListEnd . . . 529
4.79.25 TWriter.WriteSignature . . . 530
4.79.26 TWriter.WriteRootComponent . . . 530
4.79.27 TWriter.WriteString . . . 530
4.79.28 TWriter.WriteWideString . . . 530
4.79.29 TWriter.WriteUnicodeString . . . 530
4.79.30 TWriter.WriteVariant . . . 531
4.79.31 TWriter.RootAncestor . . . 531
4.79.32 TWriter.OnFindAncestor . . . 531
4.79.33 TWriter.OnWriteMethodProperty . . . 531
4.79.34 TWriter.OnWriteStringProperty . . . 532
4.79.35 TWriter.Driver . . . 532
4.79.36 TWriter.PropertyPath . . . 532

5 Reference for unit 'clocale' 533
5.1 Used units . . . 533
5.2 Overview . . . 533

6 Reference for unit 'cmem' 534
6.1 Used units . . . 534
6.2 Overview . . . 534
6.3 Constants, types and variables . . . 534
6.3.1 Constants . . . 534
6.4 Procedures and functions . . . 534
6.4.1 CAlloc . . . 534
6.4.2 Free . . . 535
6.4.3 Malloc . . . 535
6.4.4 ReAlloc . . . 535

7 Reference for unit 'collation_de' 536
7.1 Used units . . . 536
7.2 Overview . . . 536

8 Reference for unit 'collation_es' 537
8.1 Used units . . . 537
8.2 Overview . . . 537

9 Reference for unit 'collation_fr_ca' 538
9.1 Used units . . . 538

9.2 Overview . . . 538

10 Reference for unit 'collation_ja' 539
10.1 Used units . . . 539
10.2 Overview . . . 539

11 Reference for unit 'collation_ko' 540
11.1 Used units . . . 540
11.2 Overview . . . 540

12 Reference for unit 'collation_ru' 541
12.1 Used units . . . 541
12.2 Overview . . . 541

13 Reference for unit 'collation_sv' 542
13.1 Used units . . . 542
13.2 Overview . . . 542

14 Reference for unit 'collation_zh' 543
14.1 Used units . . . 543
14.2 Overview . . . 543

15 Reference for unit 'cp1250' 544
15.1 Used units . . . 544
15.2 Overview . . . 544

16 Reference for unit 'cp1251' 545
16.1 Used units . . . 545
16.2 Overview . . . 545

17 Reference for unit 'cp1252' 546
17.1 Used units . . . 546
17.2 Overview . . . 546

18 Reference for unit 'cp1253' 547
18.1 Used units . . . 547
18.2 Overview . . . 547

19 Reference for unit 'cp1254' 548
19.1 Used units . . . 548
19.2 Overview . . . 548

20 Reference for unit 'cp1255' 549
20.1 Used units . . . 549

20.2 Overview 549

21 Reference for unit 'cp1256' 550
21.1 Used units 550
21.2 Overview 550

22 Reference for unit 'cp1257' 551
22.1 Used units 551
22.2 Overview 551

23 Reference for unit 'cp1258' 552
23.1 Used units 552
23.2 Overview 552

24 Reference for unit 'cp437' 553
24.1 Used units 553
24.2 Overview 553

25 Reference for unit 'cp646' 554
25.1 Used units 554
25.2 Overview 554

26 Reference for unit 'cp850' 555
26.1 Used units 555
26.2 Overview 555

27 Reference for unit 'cp852' 556
27.1 Used units 556
27.2 Overview 556

28 Reference for unit 'cp856' 557
28.1 Used units 557
28.2 Overview 557

29 Reference for unit 'cp866' 558
29.1 Used units 558
29.2 Overview 558

30 Reference for unit 'cp874' 559
30.1 Used units 559
30.2 Overview 559

31 Reference for unit 'cp8859_1' 560
31.1 Used units 560

31.2 Overview 560

32 Reference for unit 'cp8859_2' 561
32.1 Used units 561
32.2 Overview 561

33 Reference for unit 'cp8859_5' 562
33.1 Used units 562
33.2 Overview 562

34 Reference for unit 'cp895' 563
34.1 Used units 563
34.2 Overview 563

35 Reference for unit 'cp932' 564
35.1 Used units 564
35.2 Overview 564

36 Reference for unit 'cp936' 565
36.1 Used units 565
36.2 Overview 565

37 Reference for unit 'cp949' 566
37.1 Used units 566
37.2 Overview 566

38 Reference for unit 'cp950' 567
38.1 Used units 567
38.2 Overview 567

39 Reference for unit 'cpall' 568
39.1 Used units 569
39.2 Overview 570

40 Reference for unit 'Crt' 571
40.1 Used units 571
40.2 Overview 571
40.3 Constants, types and variables 572
40.3.1 Constants 572
40.3.2 Types 574
40.3.3 Variables 574
40.4 Procedures and functions 575
40.4.1 AssignCrt 575

40.4.2 ClrEol . . . 576
40.4.3 ClrScr . . . 577
40.4.4 cursorbig . . . 577
40.4.5 cursoroff . . . 577
40.4.6 cursoron . . . 578
40.4.7 Delay . . . 578
40.4.8 DelLine . . . 578
40.4.9 GotoXY . . . 579
40.4.10 HighVideo . . . 580
40.4.11 InsLine . . . 580
40.4.12 KeyPressed . . . 581
40.4.13 LowVideo . . . 581
40.4.14 NormVideo . . . 582
40.4.15 NoSound . . . 582
40.4.16 ReadKey . . . 582
40.4.17 Sound . . . 583
40.4.18 TextBackground . . . 583
40.4.19 TextColor . . . 584
40.4.20 TextMode . . . 585
40.4.21 WhereX . . . 585
40.4.22 WhereY . . . 585
40.4.23 Window . . . 586
40.5 TCharAttr . . . 586

41 Reference for unit 'cthreads' 588
41.1 Used units . . . 588
41.2 Overview . . . 588
41.3 Procedures and functions . . . 589
41.3.1 SetCThreadManager . . . 589

42 Reference for unit 'ctypes' 590
42.1 Used units . . . 590
42.2 Overview . . . 590
42.3 Constants, types and variables . . . 590
42.3.1 Types . . . 590

43 Reference for unit 'cwstring' 596
43.1 Used units . . . 596
43.2 Overview . . . 596
43.3 Procedures and functions . . . 596
43.3.1 SetCWidestringManager . . . 596

44 Reference for unit 'DateUtils' 598
44.1 Used units . . . 598
44.2 Overview . . . 598
44.3 Constants, types and variables . . . 598
44.3.1 Constants . . . 598
44.4 Procedures and functions . . . 601
44.4.1 CompareDate . . . 601
44.4.2 CompareDateTime . . . 602
44.4.3 CompareTime . . . 603
44.4.4 DateInRange . . . 604
44.4.5 DateOf . . . 604
44.4.6 DateTimeInRange . . . 605
44.4.7 DateTimeToDosDateTime . . . 605
44.4.8 DateTimeToJulianDate . . . 606
44.4.9 DateTimeToMac . . . 606
44.4.10 DateTimeToModifiedJulianDate . . . 606
44.4.11 DateTimeToUnix . . . 606
44.4.12 DateToISO8601 . . . 607
44.4.13 DayOf . . . 607
44.4.14 DayOfTheMonth . . . 608
44.4.15 DayOfTheWeek . . . 608
44.4.16 DayOfTheYear . . . 608
44.4.17 DaysBetween . . . 609
44.4.18 DaysInAMonth . . . 610
44.4.19 DaysInAYear . . . 610
44.4.20 DaysInMonth . . . 611
44.4.21 DaysInYear . . . 611
44.4.22 DaySpan . . . 612
44.4.23 DecodeDateDay . . . 612
44.4.24 DecodeDateMonthWeek . . . 613
44.4.25 DecodeDateTime . . . 614
44.4.26 DecodeDateWeek . . . 614
44.4.27 DecodeDayOfWeekInMonth . . . 615
44.4.28 DosDateTimeToDateTime . . . 615
44.4.29 EncodeDateDay . . . 616
44.4.30 EncodeDateMonthWeek . . . 616
44.4.31 EncodeDateTime . . . 616
44.4.32 EncodeDateWeek . . . 617
44.4.33 EncodeDayOfWeekInMonth . . . 617
44.4.34 EncodeTimeInterval . . . 617

44.4.35 EndOfADay . . . 618
44.4.36 EndOfAMonth . . . 618
44.4.37 EndOfAWeek . . . 619
44.4.38 EndOfAYear . . . 620
44.4.39 EndOfTheDay . . . 620
44.4.40 EndOfTheMonth . . . 621
44.4.41 EndOfTheWeek . . . 621
44.4.42 EndOfTheYear . . . 622
44.4.43 HourOf . . . 622
44.4.44 HourOfTheDay . . . 623
44.4.45 HourOfTheMonth . . . 623
44.4.46 HourOfTheWeek . . . 623
44.4.47 HourOfTheYear . . . 624
44.4.48 HoursBetween . . . 624
44.4.49 HourSpan . . . 625
44.4.50 IncDay . . . 626
44.4.51 IncHour . . . 626
44.4.52 IncMilliSecond . . . 627
44.4.53 IncMinute . . . 627
44.4.54 IncSecond . . . 628
44.4.55 IncWeek . . . 628
44.4.56 IncYear . . . 629
44.4.57 InvalidDateDayError . . . 629
44.4.58 InvalidDateMonthWeekError . . . 630
44.4.59 InvalidDateTimeError . . . 630
44.4.60 InvalidDateWeekError . . . 630
44.4.61 InvalidDayOfWeekInMonthError . . . 631
44.4.62 IsAM . . . 631
44.4.63 IsInLeapYear . . . 631
44.4.64 ISO8601ToDate . . . 632
44.4.65 ISO8601ToDateDef . . . 632
44.4.66 IsPM . . . 633
44.4.67 IsSameDay . . . 633
44.4.68 IsSameMonth . . . 634
44.4.69 IsToday . . . 634
44.4.70 IsValidDate . . . 635
44.4.71 IsValidDateDay . . . 635
44.4.72 IsValidDateMonthWeek . . . 636
44.4.73 IsValidDateTime . . . 637
44.4.74 IsValidDateWeek . . . 638

44.4.75 IsValidTime . . . 638
44.4.76 JulianDateToDateTime . . . 639
44.4.77 LocalTimeToUniversal . . . 639
44.4.78 MacTimeStampToUnix . . . 639
44.4.79 MacToDateTime . . . 639
44.4.80 MilliSecondOf . . . 640
44.4.81 MilliSecondOfTheDay . . . 640
44.4.82 MilliSecondOfTheHour . . . 640
44.4.83 MilliSecondOfTheMinute . . . 641
44.4.84 MilliSecondOfTheMonth . . . 641
44.4.85 MilliSecondOfTheSecond . . . 641
44.4.86 MilliSecondOfTheWeek . . . 642
44.4.87 MilliSecondOfTheYear . . . 642
44.4.88 MilliSecondsBetween . . . 642
44.4.89 MilliSecondSpan . . . 643
44.4.90 MinuteOf . . . 644
44.4.91 MinuteOfTheDay . . . 644
44.4.92 MinuteOfTheHour . . . 644
44.4.93 MinuteOfTheMonth . . . 645
44.4.94 MinuteOfTheWeek . . . 645
44.4.95 MinuteOfTheYear . . . 646
44.4.96 MinutesBetween . . . 646
44.4.97 MinuteSpan . . . 647
44.4.98 ModifiedJulianDateToDateTime . . . 648
44.4.99 MonthOf . . . 648
44.4.100 MonthOfTheYear . . . 648
44.4.101 MonthsBetween . . . 649
44.4.102 MonthSpan . . . 649
44.4.103 NthDayOfWeek . . . 650
44.4.104 PeriodBetween . . . 651
44.4.105 PreviousDayOfWeek . . . 651
44.4.106 RecodeDate . . . 652
44.4.107 RecodeDateTime . . . 652
44.4.108 RecodeDay . . . 653
44.4.109 RecodeHour . . . 654
44.4.110 RecodeMilliSecond . . . 654
44.4.111 RecodeMinute . . . 655
44.4.112 RecodeMonth . . . 656
44.4.113 RecodeSecond . . . 656
44.4.114 RecodeTime . . . 657

44.4.115 RecodeYear . . . 658
44.4.116 SameDate . . . 658
44.4.117 SameDateTime . . . 659
44.4.118 SameTime . . . 660
44.4.119 ScanDateTime . . . 661
44.4.120 SecondOf . . . 661
44.4.121 SecondOfTheDay . . . 662
44.4.122 SecondOfTheHour . . . 662
44.4.123 SecondOfTheMinute . . . 662
44.4.124 SecondOfTheMonth . . . 663
44.4.125 SecondOfTheWeek . . . 663
44.4.126 SecondOfTheYear . . . 663
44.4.127 SecondsBetween . . . 664
44.4.128 SecondSpan . . . 664
44.4.129 StartOfADay . . . 665
44.4.130 StartOfAMonth . . . 666
44.4.131 StartOfAWeek . . . 667
44.4.132 StartOfAYear . . . 667
44.4.133 StartOfTheDay . . . 668
44.4.134 StartOfTheMonth . . . 668
44.4.135 StartOfTheWeek . . . 669
44.4.136 StartOfTheYear . . . 669
44.4.137 TimeInRange . . . 670
44.4.138 TimeOf . . . 670
44.4.139 Today . . . 671
44.4.140 Tomorrow . . . 671
44.4.141 TryEncodeDateDay . . . 671
44.4.142 TryEncodeDateMonthWeek . . . 672
44.4.143 TryEncodeDateTime . . . 673
44.4.144 TryEncodeDateWeek . . . 674
44.4.145 TryEncodeDayOfWeekInMonth . . . 674
44.4.146 TryEncodeTimeInterval . . . 675
44.4.147 TryISO8601ToDate . . . 675
44.4.148 TryISOStrToDate . . . 676
44.4.149 TryISOStrToDateTime . . . 676
44.4.150 TryISOStrToTime . . . 677
44.4.151 TryISOTZStrToTZOffset . . . 678
44.4.152 TryJulianDateToDateTime . . . 678
44.4.153 TryModifiedJulianDateToDateTime . . . 678
44.4.154 TryRecodeDateTime . . . 679

44.4.155 UniversalTimeToLocal . . . 680
44.4.156 UnixTimeStampToMac . . . 680
44.4.157 UnixToDateTime . . . 680
44.4.158 WeekOf . . . 680
44.4.159 WeekOfTheMonth . . . 681
44.4.160 WeekOfTheYear . . . 681
44.4.161 WeeksBetween . . . 682
44.4.162 WeeksInAYear . . . 683
44.4.163 WeeksInYear . . . 684
44.4.164 WeekSpan . . . 684
44.4.165 WithinPastDays . . . 685
44.4.166 WithinPastHours . . . 686
44.4.167 WithinPastMilliSeconds . . . 687
44.4.168 WithinPastMinutes . . . 688
44.4.169 WithinPastMonths . . . 689
44.4.170 WithinPastSeconds . . . 690
44.4.171 WithinPastWeeks . . . 691
44.4.172 WithinPastYears . . . 692
44.4.173 YearOf . . . 693
44.4.174 YearsBetween . . . 693
44.4.175 YearSpan . . . 694
44.4.176 Yesterday . . . 695
44.5 TDateTimeHelper . . . 696
44.5.1 Method overview . . . 696
44.5.2 Property overview . . . 697
44.5.3 TDateTimeHelper.Create . . . 697
44.5.4 TDateTimeHelper.CreateLocal . . . 697
44.5.5 TDateTimeHelper.CreateUnixTime . . . 697
44.5.6 TDateTimeHelper.CreateTotalSeconds . . . 697
44.5.7 TDateTimeHelper.ToString . . . 698
44.5.8 TDateTimeHelper.StartOfYear . . . 698
44.5.9 TDateTimeHelper.EndOfYear . . . 698
44.5.10 TDateTimeHelper.StartOfMonth . . . 698
44.5.11 TDateTimeHelper.EndOfMonth . . . 698
44.5.12 TDateTimeHelper.StartOfWeek . . . 698
44.5.13 TDateTimeHelper.EndOfWeek . . . 698
44.5.14 TDateTimeHelper.StartOfDay . . . 698
44.5.15 TDateTimeHelper.EndOfDay . . . 699
44.5.16 TDateTimeHelper.AddYears . . . 699
44.5.17 TDateTimeHelper.AddMonths . . . 699

44.5.18 TDateTimeHelper.AddDays . . . 699
44.5.19 TDateTimeHelper.AddHours . . . 699
44.5.20 TDateTimeHelper.AddMinutes . . . 699
44.5.21 TDateTimeHelper.AddSeconds . . . 699
44.5.22 TDateTimeHelper.AddMilliseconds . . . 699
44.5.23 TDateTimeHelper.CompareTo . . . 699
44.5.24 TDateTimeHelper.Equals . . . 700
44.5.25 TDateTimeHelper.IsSameDay . . . 700
44.5.26 TDateTimeHelper.InRange . . . 700
44.5.27 TDateTimeHelper.IsInLeapYear . . . 700
44.5.28 TDateTimeHelper.IsToday . . . 700
44.5.29 TDateTimeHelper.IsAM . . . 700
44.5.30 TDateTimeHelper.IsPM . . . 700
44.5.31 TDateTimeHelper.YearsBetween . . . 700
44.5.32 TDateTimeHelper.MonthsBetween . . . 701
44.5.33 TDateTimeHelper.WeeksBetween . . . 701
44.5.34 TDateTimeHelper.DaysBetween . . . 701
44.5.35 TDateTimeHelper.HoursBetween . . . 701
44.5.36 TDateTimeHelper.MinutesBetween . . . 701
44.5.37 TDateTimeHelper.SecondsBetween . . . 701
44.5.38 TDateTimeHelper.MilliSecondsBetween . . . 701
44.5.39 TDateTimeHelper.WithinYears . . . 701
44.5.40 TDateTimeHelper.WithinMonths . . . 701
44.5.41 TDateTimeHelper.WithinWeeks . . . 702
44.5.42 TDateTimeHelper.WithinDays . . . 702
44.5.43 TDateTimeHelper.WithinHours . . . 702
44.5.44 TDateTimeHelper.WithinMinutes . . . 702
44.5.45 TDateTimeHelper.WithinSeconds . . . 702
44.5.46 TDateTimeHelper.WithinMilliseconds . . . 702
44.5.47 TDateTimeHelper.Now . . . 702
44.5.48 TDateTimeHelper.Today . . . 703
44.5.49 TDateTimeHelper.Yesterday . . . 703
44.5.50 TDateTimeHelper.Tomorrow . . . 703
44.5.51 TDateTimeHelper.Date . . . 703
44.5.52 TDateTimeHelper.Time . . . 703
44.5.53 TDateTimeHelper.DayOfWeek . . . 703
44.5.54 TDateTimeHelper.DayOfYear . . . 703
44.5.55 TDateTimeHelper.Year . . . 704
44.5.56 TDateTimeHelper.Month . . . 704
44.5.57 TDateTimeHelper.Day . . . 704

44.5.58 TDateTimeHelper.Hour . . . 704
44.5.59 TDateTimeHelper.Minute . . . 704
44.5.60 TDateTimeHelper.Second . . . 704
44.5.61 TDateTimeHelper.Millisecond . . . 704
44.5.62 TDateTimeHelper.UnixTime . . . 705
44.5.63 TDateTimeHelper.TotalSeconds . . . 705

45 Reference for unit 'Dos' 706
45.1 Used units . . . 706
45.2 Overview . . . 706
45.3 System information. . . . 706
45.4 Process handling. . . . 707
45.5 Directory and disk handling. . . . 707
45.6 File handling. . . . 708
45.7 File open mode constants. . . . 708
45.8 File attributes. . . . 708
45.9 Constants, types and variables . . . 709
45.9.1 Constants . . . 709
45.9.2 Types . . . 710
45.9.3 Variables . . . 712
45.10 Procedures and functions . . . 712
45.10.1 AddDisk . . . 712
45.10.2 DiskFree . . . 713
45.10.3 DiskSize . . . 714
45.10.4 DosExitCode . . . 714
45.10.5 DosVersion . . . 715
45.10.6 DTToUnixDate . . . 715
45.10.7 EnvCount . . . 716
45.10.8 EnvStr . . . 716
45.10.9 Exec . . . 717
45.10.10 FExpand . . . 717
45.10.11 FindClose . . . 717
45.10.12 FindFirst . . . 718
45.10.13 FindNext . . . 719
45.10.14 FSearch . . . 719
45.10.15 FSplit . . . 720
45.10.16 GetCBreak . . . 720
45.10.17 GetDate . . . 721
45.10.18 GetEnv . . . 721
45.10.19 GetFAttr . . . 722

45.10.20 GetFTime . . . 723
45.10.21 GetIntVec . . . 723
45.10.22 GetLongName . . . 724
45.10.23 GetMsCount . . . 724
45.10.24 GetShortName . . . 724
45.10.25 GetTime . . . 725
45.10.26 GetVerify . . . 725
45.10.27 Intr . . . 726
45.10.28 Keep . . . 726
45.10.29 MSDos . . . 726
45.10.30 PackTime . . . 727
45.10.31 SetCBreak . . . 727
45.10.32 SetDate . . . 728
45.10.33 SetFAttr . . . 728
45.10.34 SetFTime . . . 729
45.10.35 SetIntVec . . . 729
45.10.36 SetTime . . . 729
45.10.37 SetVerify . . . 729
45.10.38 SwapVectors . . . 730
45.10.39 UnixDateToDt . . . 730
45.10.40 UnpackTime . . . 730
45.10.41 weekday . . . 731
45.11 DateTime . . . 731
45.12 SearchRec . . . 731

46 Reference for unit 'dxeload' 733
46.1 Used units . . . 733
46.2 Overview . . . 733
46.3 Procedures and functions . . . 733
46.3.1 dxe_load . . . 733

47 Reference for unit 'dynlibs' 734
47.1 Used units . . . 734
47.2 Overview . . . 734
47.3 Constants, types and variables . . . 734
47.3.1 Constants . . . 734
47.3.2 Types . . . 734
47.4 Procedures and functions . . . 735
47.4.1 FreeLibrary . . . 735
47.4.2 GetLoadErrorStr . . . 735
47.4.3 GetProcAddress . . . 735

47.4.4 GetProcedureAddress . . . 736
47.4.5 LoadLibrary . . . 736
47.4.6 SafeLoadLibrary . . . 736
47.4.7 UnloadLibrary . . . 737

48 Reference for unit 'emu387' 738
48.1 Used units . . . 738
48.2 Overview . . . 738
48.3 Procedures and functions . . . 738
48.3.1 npxsetup . . . 738

49 Reference for unit 'errors' 740
49.1 Used units . . . 740
49.2 Overview . . . 740
49.3 Constants, types and variables . . . 740
49.3.1 Constants . . . 740
49.4 Procedures and functions . . . 741
49.4.1 PError . . . 741
49.4.2 StrError . . . 742

50 Reference for unit 'exeinfo' 743
50.1 Used units . . . 743
50.2 Overview . . . 743
50.3 Constants, types and variables . . . 743
50.3.1 Types . . . 743
50.4 Procedures and functions . . . 743
50.4.1 CloseExeFile . . . 743
50.4.2 FindExeSection . . . 744
50.4.3 GetModuleByAddr . . . 744
50.4.4 OpenExeFile . . . 744
50.4.5 ReadDebugLink . . . 745
50.5 TExeFile . . . 745

51 Reference for unit 'fgl' 746
51.1 Used units . . . 746
51.2 Overview . . . 746
51.3 Constants, types and variables . . . 746
51.3.1 Constants . . . 746
51.3.2 Types . . . 746
51.4 EListError . . . 747
51.4.1 Description . . . 747

51.5 TFPGInterfacedObjectList . . . 747
51.5.1 Description . . . 747
51.5.2 Method overview . . . 747
51.5.3 Property overview . . . 747
51.5.4 TFPGInterfacedObjectList.Create . . . 748
51.5.5 TFPGInterfacedObjectList.Add . . . 748
51.5.6 TFPGInterfacedObjectList.Extract . . . 748
51.5.7 TFPGInterfacedObjectList.GetEnumerator . . . 748
51.5.8 TFPGInterfacedObjectList.IndexOf . . . 749
51.5.9 TFPGInterfacedObjectList.Insert . . . 749
51.5.10 TFPGInterfacedObjectList.Assign . . . 749
51.5.11 TFPGInterfacedObjectList.AddList . . . 749
51.5.12 TFPGInterfacedObjectList.Remove . . . 750
51.5.13 TFPGInterfacedObjectList.Sort . . . 750
51.5.14 TFPGInterfacedObjectList.First . . . 750
51.5.15 TFPGInterfacedObjectList.Last . . . 751
51.5.16 TFPGInterfacedObjectList.Items . . . 751
51.5.17 TFPGInterfacedObjectList.List . . . 751
51.6 TFPGList . . . 751
51.6.1 Description . . . 751
51.6.2 Method overview . . . 752
51.6.3 Property overview . . . 752
51.6.4 TFPGList.Create . . . 752
51.6.5 TFPGList.ItemIsManaged . . . 752
51.6.6 TFPGList.Add . . . 752
51.6.7 TFPGList.Extract . . . 753
51.6.8 TFPGList.GetEnumerator . . . 753
51.6.9 TFPGList.IndexOf . . . 753
51.6.10 TFPGList.Insert . . . 753
51.6.11 TFPGList.Assign . . . 754
51.6.12 TFPGList.AddList . . . 754
51.6.13 TFPGList.Remove . . . 754
51.6.14 TFPGList.Sort . . . 754
51.6.15 TFPGList.First . . . 755
51.6.16 TFPGList.Last . . . 755
51.6.17 TFPGList.Items . . . 755
51.6.18 TFPGList.List . . . 755
51.7 TFPGListEnumerator . . . 756
51.7.1 Description . . . 756
51.7.2 Method overview . . . 756

51.7.3 Property overview . . . 756
51.7.4 TFPGListEnumerator.Create . . . 756
51.7.5 TFPGListEnumerator.MoveNext . . . 756
51.7.6 TFPGListEnumerator.Current . . . 756
51.8 TFPGMap . . . 757
51.8.1 Description . . . 757
51.8.2 Method overview . . . 757
51.8.3 Property overview . . . 757
51.8.4 TFPGMap.Create . . . 757
51.8.5 TFPGMap.Add . . . 758
51.8.6 TFPGMap.Find . . . 758
51.8.7 TFPGMap.TryGetData . . . 758
51.8.8 TFPGMap.AddOrSetData . . . 758
51.8.9 TFPGMap.IndexOf . . . 759
51.8.10 TFPGMap.IndexOfData . . . 759
51.8.11 TFPGMap.InsertKey . . . 759
51.8.12 TFPGMap.InsertKeyData . . . 759
51.8.13 TFPGMap.Remove . . . 760
51.8.14 TFPGMap.Keys . . . 760
51.8.15 TFPGMap.Data . . . 760
51.8.16 TFPGMap.KeyData . . . 761
51.8.17 TFPGMap.OnCompare . . . 761
51.8.18 TFPGMap.OnKeyCompare . . . 761
51.8.19 TFPGMap.OnDataCompare . . . 762
51.9 TFPGMapInterfacedObjectData . . . 762
51.9.1 Description . . . 762
51.9.2 Method overview . . . 762
51.9.3 Property overview . . . 763
51.9.4 TFPGMapInterfacedObjectData.Create . . . 763
51.9.5 TFPGMapInterfacedObjectData.Add . . . 763
51.9.6 TFPGMapInterfacedObjectData.Find . . . 763
51.9.7 TFPGMapInterfacedObjectData.TryGetData . . . 764
51.9.8 TFPGMapInterfacedObjectData.AddOrSetData . . . 764
51.9.9 TFPGMapInterfacedObjectData.IndexOf . . . 764
51.9.10 TFPGMapInterfacedObjectData.IndexOfData . . . 764
51.9.11 TFPGMapInterfacedObjectData.InsertKey . . . 765
51.9.12 TFPGMapInterfacedObjectData.InsertKeyData . . . 765
51.9.13 TFPGMapInterfacedObjectData.Remove . . . 765
51.9.14 TFPGMapInterfacedObjectData.Keys . . . 765
51.9.15 TFPGMapInterfacedObjectData.Data . . . 766

51.9.16 TFPGMapInterfacedObjectData.KeyData . . . 766
51.9.17 TFPGMapInterfacedObjectData.OnCompare . . . 766
51.9.18 TFPGMapInterfacedObjectData.OnKeyCompare . . . 767
51.9.19 TFPGMapInterfacedObjectData.OnDataCompare . . . 767
51.10 TFPGMapObject . . . 767
51.10.1 Description . . . 767
51.10.2 Method overview . . . 768
51.10.3 Property overview . . . 768
51.10.4 TFPGMapObject.Create . . . 768
51.10.5 TFPGMapObject.Add . . . 768
51.10.6 TFPGMapObject.Find . . . 769
51.10.7 TFPGMapObject.TryGetData . . . 769
51.10.8 TFPGMapObject.AddOrSetData . . . 769
51.10.9 TFPGMapObject.IndexOf . . . 769
51.10.10 TFPGMapObject.IndexOfData . . . 770
51.10.11 TFPGMapObject.InsertKey . . . 770
51.10.12 TFPGMapObject.InsertKeyData . . . 770
51.10.13 TFPGMapObject.Remove . . . 770
51.10.14 TFPGMapObject.Keys . . . 771
51.10.15 TFPGMapObject.Data . . . 771
51.10.16 TFPGMapObject.KeyData . . . 771
51.10.17 TFPGMapObject.OnCompare . . . 772
51.10.18 TFPGMapObject.OnKeyCompare . . . 772
51.10.19 TFPGMapObject.OnDataCompare . . . 772
51.11 TFPGObjectList . . . 773
51.11.1 Description . . . 773
51.11.2 Method overview . . . 773
51.11.3 Property overview . . . 773
51.11.4 TFPGObjectList.Create . . . 773
51.11.5 TFPGObjectList.Add . . . 773
51.11.6 TFPGObjectList.Extract . . . 774
51.11.7 TFPGObjectList.GetEnumerator . . . 774
51.11.8 TFPGObjectList.IndexOf . . . 774
51.11.9 TFPGObjectList.Insert . . . 775
51.11.10 TFPGObjectList.AddList . . . 775
51.11.11 TFPGObjectList.Assign . . . 775
51.11.12 TFPGObjectList.Remove . . . 775
51.11.13 TFPGObjectList.Sort . . . 776
51.11.14 TFPGObjectList.First . . . 776
51.11.15 TFPGObjectList.Last . . . 776

51.11.16 TFPGObjectList.Items . . . 776
51.11.17 TFPGObjectList.List . . . 777
51.11.18 TFPGObjectList.FreeObjects . . . 777
51.12 TFPSList . . . 777
51.12.1 Description . . . 777
51.12.2 Method overview . . . 778
51.12.3 Property overview . . . 778
51.12.4 TFPSList.Create . . . 778
51.12.5 TFPSList.Destroy . . . 778
51.12.6 TFPSList.ItemIsManaged . . . 779
51.12.7 TFPSList.Add . . . 779
51.12.8 TFPSList.Clear . . . 779
51.12.9 TFPSList.Delete . . . 779
51.12.10 TFPSList.DeleteRange . . . 780
51.12.11 TFPSList.Error . . . 780
51.12.12 TFPSList.Exchange . . . 780
51.12.13 TFPSList.Expand . . . 780
51.12.14 TFPSList.Extract . . . 781
51.12.15 TFPSList.IndexOf . . . 781
51.12.16 TFPSList.Insert . . . 781
51.12.17 TFPSList.Move . . . 782
51.12.18 TFPSList.Assign . . . 782
51.12.19 TFPSList.AddList . . . 782
51.12.20 TFPSList.Remove . . . 782
51.12.21 TFPSList.Pack . . . 783
51.12.22 TFPSList.Sort . . . 783
51.12.23 TFPSList.Capacity . . . 783
51.12.24 TFPSList.Count . . . 784
51.12.25 TFPSList.Items . . . 784
51.12.26 TFPSList.ItemSize . . . 784
51.12.27 TFPSList.List . . . 784
51.12.28 TFPSList.First . . . 785
51.12.29 TFPSList.Last . . . 785
51.13 TFPSMap . . . 785
51.13.1 Description . . . 785
51.13.2 Method overview . . . 785
51.13.3 Property overview . . . 786
51.13.4 TFPSMap.Create . . . 786
51.13.5 TFPSMap.Add . . . 786
51.13.6 TFPSMap.Find . . . 786

51.13.7 TFPSMap.IndexOf . . . 787
51.13.8 TFPSMap.IndexOfData . . . 787
51.13.9 TFPSMap.Insert . . . 787
51.13.10 TFPSMap.InsertKey . . . 788
51.13.11 TFPSMap.InsertKeyData . . . 788
51.13.12 TFPSMap.Remove . . . 788
51.13.13 TFPSMap.Sort . . . 788
51.13.14 TFPSMap.Duplicates . . . 789
51.13.15 TFPSMap.KeySize . . . 789
51.13.16 TFPSMap.DataSize . . . 789
51.13.17 TFPSMap.Keys . . . 790
51.13.18 TFPSMap.Data . . . 790
51.13.19 TFPSMap.KeyData . . . 790
51.13.20 TFPSMap.Sorted . . . 790
51.13.21 TFPSMap.OnPtrCompare . . . 791
51.13.22 TFPSMap.OnKeyPtrCompare . . . 791
51.13.23 TFPSMap.OnDataPtrCompare . . . 791

52 Reference for unit 'fpwidestring' 792
52.1 Used units . . . 792
52.2 Overview . . . 792
52.3 Constants, types and variables . . . 793
52.3.1 Variables . . . 793
52.4 Procedures and functions . . . 793
52.4.1 GetActiveCollation . . . 793
52.4.2 SetActiveCollation . . . 793

53 Reference for unit 'getopts' 794
53.1 Used units . . . 794
53.2 Overview . . . 794
53.3 Constants, types and variables . . . 794
53.3.1 Constants . . . 794
53.3.2 Types . . . 795
53.3.3 Variables . . . 795
53.4 Procedures and functions . . . 795
53.4.1 GetLongOpts . . . 795
53.4.2 GetOpt . . . 796
53.5 TOption . . . 798
53.5.1 Method overview . . . 799
53.5.2 TOption.SetOption . . . 799

54 Reference for unit 'go32' 800
54.1 Used units . . . 800
54.2 Overview . . . 800
54.3 Real mode callbacks. . . . 800
54.4 Executing software interrupts. . . . 801
54.5 Software interrupts. . . . 804
54.6 Hardware interrupts. . . . 804
54.7 Disabling interrupts. . . . 806
54.8 Creating your own interrupt handlers. . . . 806
54.9 Protected mode interrupts vs. Real mode interrupts. . . . 807
54.10 Handling interrupts with DPMI. . . . 807
54.11 Interrupt redirection. . . . 807
54.12 Processor access. . . . 807
54.13 I/O port access. . . . 807
54.14 dos memory access. . . . 807
54.15 FPC specialities. . . . 808
54.16 Selectors and descriptors. . . . 808
54.17 What is DPMI. . . . 808
54.18 Constants, types and variables . . . 808
54.18.1 Constants . . . 808
54.18.2 Types . . . 811
54.18.3 Variables . . . 813
54.19 Procedures and functions . . . 813
54.19.1 allocate_ldt_descriptors . . . 813
54.19.2 allocate_memory_block . . . 816
54.19.3 copyfromdos . . . 817
54.19.4 copytodos . . . 817
54.19.5 create_code_segment_alias_descriptor . . . 818
54.19.6 disable . . . 818
54.19.7 dpmi_dosmemfillchar . . . 818
54.19.8 dpmi_dosmemfillword . . . 819
54.19.9 dpmi_dosmemget . . . 819
54.19.10 dpmi_dosmemmove . . . 819
54.19.11 dpmi_dosmemput . . . 819
54.19.12 enable . . . 820
54.19.13 free_ldt_descriptor . . . 820
54.19.14 free_linear_addr_mapping . . . 820
54.19.15 free_memory_block . . . 820
54.19.16 free_rm_callback . . . 821
54.19.17 get_cs . . . 821

54.19.18 get_descriptor_access_right . . . 821
54.19.19 get_dpmi_version . . . 822
54.19.20 get_ds . . . 822
54.19.21 get_exception_handler . . . 822
54.19.22 get_linear_addr . . . 823
54.19.23 get_meminfo . . . 823
54.19.24 get_next_selector_increment_value . . . 824
54.19.25 get_page_attributes . . . 825
54.19.26 get_page_size . . . 825
54.19.27 get_pm_exception_handler . . . 825
54.19.28 get_pm_interrupt . . . 825
54.19.29 get_rm_callback . . . 826
54.19.30 get_rm_interrupt . . . 830
54.19.31 get_run_mode . . . 831
54.19.32 get_segment_base_address . . . 832
54.19.33 get_segment_limit . . . 832
54.19.34 get_ss . . . 832
54.19.35 global_dos_alloc . . . 833
54.19.36 global_dos_free . . . 835
54.19.37 inportb . . . 836
54.19.38 inportl . . . 836
54.19.39 inportw . . . 836
54.19.40 lock_code . . . 837
54.19.41 lock_data . . . 837
54.19.42 lock_linear_region . . . 837
54.19.43 map_device_in_memory_block . . . 838
54.19.44 outportb . . . 838
54.19.45 outportl . . . 839
54.19.46 outportw . . . 839
54.19.47 realintr . . . 839
54.19.48 request_linear_region . . . 840
54.19.49 segment_to_descriptor . . . 841
54.19.50 seg_fillchar . . . 841
54.19.51 seg_fillword . . . 842
54.19.52 seg_move . . . 843
54.19.53 set_descriptor_access_right . . . 843
54.19.54 set_exception_handler . . . 843
54.19.55 set_page_attributes . . . 844
54.19.56 set_pm_exception_handler . . . 844
54.19.57 set_pm_interrupt . . . 844

54.19.58 set_rm_interrupt . . . 846
54.19.59 set_segment_base_address . . . 846
54.19.60 set_segment_limit . . . 846
54.19.61 tb_offset . . . 847
54.19.62 tb_segment . . . 847
54.19.63 tb_size . . . 847
54.19.64 transfer_buffer . . . 848
54.19.65 unlock_code . . . 848
54.19.66 unlock_data . . . 848
54.19.67 unlock_linear_region . . . 849
54.20 tdpmiversioninfo . . . 849
54.21 tmeminfo . . . 850
54.22 tseginfo . . . 850

55 Reference for unit 'gpm' 851
55.1 Used units . . . 851
55.2 Overview . . . 851
55.3 Constants, types and variables . . . 851
55.3.1 Constants . . . 851
55.3.2 Types . . . 853
55.3.3 Variables . . . 854
55.4 Procedures and functions . . . 855
55.4.1 Gpm_AnyDouble . . . 855
55.4.2 Gpm_AnySingle . . . 855
55.4.3 Gpm_AnyTriple . . . 855
55.4.4 gpm_close . . . 856
55.4.5 gpm_fitvalues . . . 856
55.4.6 gpm_fitvaluesM . . . 856
55.4.7 gpm_getevent . . . 857
55.4.8 gpm_getsnapshot . . . 858
55.4.9 gpm_lowerroi . . . 858
55.4.10 gpm_open . . . 858
55.4.11 gpm_poproi . . . 859
55.4.12 gpm_pushroi . . . 859
55.4.13 gpm_raiseroi . . . 859
55.4.14 gpm_repeat . . . 860
55.4.15 Gpm_StrictDouble . . . 860
55.4.16 Gpm_StrictSingle . . . 860
55.4.17 Gpm_StrictTriple . . . 860
55.5 Tgpm_connect . . . 861

55.6 Tgpm_event . . . 861
55.7 Tgpm_roi . . . 861

56 Reference for unit 'Graph' 863
56.1 Used units . . . 863
56.2 Overview . . . 863
56.3 Categorized functions: Text and font handling. . . . 863
56.4 Categorized functions: Filled drawings. . . . 864
56.5 Categorized functions: Drawing primitives. . . . 864
56.6 Categorized functions: Color management. . . . 865
56.7 Categorized functions: Screen management. . . . 865
56.8 Categorized functions: Initialization. . . . 866
56.9 Target specific issues: Linux. . . . 866
56.10 Target specific issues: DOS. . . . 867
56.11 A word about mode selection. . . . 867
56.12 Requirements. . . . 872
56.13 Constants, types and variables . . . 872
56.13.1 Constants . . . 872
56.13.2 Types . . . 889
56.13.3 Variables . . . 892
56.14 Procedures and functions . . . 895
56.14.1 Arc . . . 895
56.14.2 Bar . . . 895
56.14.3 Bar3D . . . 895
56.14.4 ClearDevice . . . 896
56.14.5 Closegraph . . . 896
56.14.6 DetectGraph . . . 896
56.14.7 DrawPoly . . . 896
56.14.8 Ellipse . . . 897
56.14.9 FillEllipse . . . 897
56.14.10 FillPoly . . . 897
56.14.11 FloodFill . . . 897
56.14.12 GetArcCoords . . . 898
56.14.13 GetAspectRatio . . . 898
56.14.14 GetColor . . . 898
56.14.15 GetDefaultPalette . . . 898
56.14.16 GetDirectVideo . . . 899
56.14.17 GetDriverName . . . 899
56.14.18 GetFillPattern . . . 899
56.14.19 GetFillSettings . . . 899

56.14.20 GetGraphMode . . . 900
56.14.21 GetLineSettings . . . 900
56.14.22 GetMaxColor . . . 900
56.14.23 GetMaxMode . . . 900
56.14.24 GetMaxX . . . 901
56.14.25 GetMaxY . . . 901
56.14.26 GetModeName . . . 901
56.14.27 GetModeRange . . . 901
56.14.28 GetPalette . . . 902
56.14.29 GetPaletteSize . . . 902
56.14.30 GetTextSettings . . . 902
56.14.31 GetViewSettings . . . 902
56.14.32 GetX . . . 903
56.14.33 GetY . . . 903
56.14.34 GraphDefaults . . . 903
56.14.35 GraphErrorMsg . . . 903
56.14.36 GraphResult . . . 904
56.14.37 InitGraph . . . 904
56.14.38 InstallUserDriver . . . 905
56.14.39 InstallUserFont . . . 905
56.14.40 LineRel . . . 905
56.14.41 LineTo . . . 906
56.14.42 MoveRel . . . 906
56.14.43 MoveTo . . . 906
56.14.44 OutText . . . 906
56.14.45 PieSlice . . . 907
56.14.46 queryadapterinfo . . . 907
56.14.47 Rectangle . . . 907
56.14.48 RegisterBGIDriver . . . 907
56.14.49 RegisterBGIfont . . . 908
56.14.50 RestoreCrtMode . . . 908
56.14.51 Sector . . . 908
56.14.52 SetAspectRatio . . . 908
56.14.53 SetColor . . . 909
56.14.54 SetDirectVideo . . . 909
56.14.55 SetFillPattern . . . 909
56.14.56 SetFillStyle . . . 909
56.14.57 SetGraphMode . . . 910
56.14.58 SetLineStyle . . . 910
56.14.59 SetPalette . . . 911

56.14.60 SetTextJustify . . . 911
56.14.61 SetTextStyle . . . 911
56.14.62 SetUserCharSize . . . 912
56.14.63 SetViewPort . . . 912
56.14.64 SetWriteMode . . . 913
56.14.65 SetWriteModeEx . . . 913
56.14.66 TextHeight . . . 913
56.14.67 TextWidth . . . 913
56.15 ArcCoordsType . . . 914
56.16 FillSettingsType . . . 914
56.17 LineSettingsType . . . 914
56.18 PaletteType . . . 914
56.19 PointType . . . 914
56.20 RGBRec . . . 915
56.21 TextSettingsType . . . 915
56.22 TModeInfo . . . 915
56.23 TResolutionRec . . . 916
56.24 ViewPortType . . . 916

57 Reference for unit 'heaptrc' 918
57.1 Used units . . . 918
57.2 Overview . . . 918
57.3 Controlling HeapTrc with environment variables. . . . 919
57.4 HeapTrc Usage. . . . 919
57.5 Constants, types and variables . . . 920
57.5.1 Constants . . . 920
57.5.2 Types . . . 922
57.6 Procedures and functions . . . 922
57.6.1 CheckPointer . . . 922
57.6.2 DumpHeap . . . 922
57.6.3 SetHeapExtraInfo . . . 923
57.6.4 SetHeapTraceOutput . . . 924

58 Reference for unit 'ipc' 925
58.1 Used units . . . 925
58.2 Overview . . . 925
58.3 Constants, types and variables . . . 925
58.3.1 Constants . . . 925
58.3.2 Types . . . 928
58.4 Procedures and functions . . . 929
58.4.1 ftok . . . 929

58.4.2 msgctl . . . 930
58.4.3 msgget . . . 932
58.4.4 msgrcv . . . 933
58.4.5 msgsnd . . . 933
58.4.6 semctl . . . 934
58.4.7 semget . . . 939
58.4.8 semop . . . 939
58.4.9 semtimedop . . . 940
58.4.10 shmat . . . 940
58.4.11 shmctl . . . 941
58.4.12 shmdt . . . 943
58.4.13 shmget . . . 943
58.5 TIPC_Perm . . . 944
58.6 TMSG . . . 944
58.7 TMSGbuf . . . 944
58.8 TMSGinfo . . . 944
58.9 TMSQid_ds . . . 945
58.10 TSEMbuf . . . 945
58.11 TSEMid_ds . . . 945
58.12 TSEMinfo . . . 946
58.13 TShmid_ds . . . 946
58.14 TSHMinfo . . . 947
58.15 TSHM_info . . . 947

59 Reference for unit 'keyboard' 948
59.1 Used units . . . 948
59.2 Overview . . . 948
59.3 Unix specific notes. . . . 948
59.4 Writing a keyboard driver. . . . 950
59.5 Keyboard scan codes. . . . 953
59.6 Constants, types and variables . . . 955
59.6.1 Constants . . . 955
59.6.2 Types . . . 960
59.7 Procedures and functions . . . 960
59.7.1 DoneKeyboard . . . 960
59.7.2 FunctionKeyName . . . 961
59.7.3 GetKeyboardDriver . . . 961
59.7.4 GetKeyEvent . . . 962
59.7.5 GetKeyEventChar . . . 962
59.7.6 GetKeyEventCode . . . 963

59.7.7 GetKeyEventFlags . . . 964
59.7.8 GetKeyEventShiftState . . . 964
59.7.9 GetKeyEventUniCode . . . 965
59.7.10 InitKeyboard . . . 965
59.7.11 IsFunctionKey . . . 965
59.7.12 KeyEventToString . . . 966
59.7.13 KeyPressed . . . 966
59.7.14 PollKeyEvent . . . 967
59.7.15 PollShiftStateEvent . . . 967
59.7.16 PutKeyEvent . . . 968
59.7.17 SetKeyboardDriver . . . 969
59.7.18 ShiftStateToString . . . 969
59.7.19 TranslateKeyEvent . . . 970
59.7.20 TranslateKeyEventUniCode . . . 970
59.8 TKeyboardDriver . . . 970
59.9 TKeyRecord . . . 971
59.10 TTreeElement . . . 971

60 Reference for unit 'lineinfo' 973
60.1 Used units . . . 973
60.2 Overview . . . 973
60.3 Constants, types and variables . . . 973
60.3.1 Types . . . 973
60.3.2 Variables . . . 973
60.4 Procedures and functions . . . 974
60.4.1 CloseStabs . . . 974
60.4.2 GetLineInfo . . . 974
60.4.3 StabBackTraceStr . . . 974

61 Reference for unit 'Linux' 975
61.1 Used units . . . 975
61.2 Overview . . . 975
61.3 Constants, types and variables . . . 975
61.3.1 Constants . . . 975
61.3.2 Types . . . 991
61.4 Procedures and functions . . . 993
61.4.1 capget . . . 993
61.4.2 capset . . . 993
61.4.3 clock_getres . . . 994
61.4.4 clock_gettime . . . 994
61.4.5 clock_settime . . . 994

61.4.6 clone . . . 994
61.4.7 epoll_create . . . 995
61.4.8 epoll_ctl . . . 996
61.4.9 epoll_wait . . . 996
61.4.10 fdatasync . . . 997
61.4.11 futex . . . 997
61.4.12 futex_op . . . 998
61.4.13 futimens . . . 998
61.4.14 inotify_add_watch . . . 998
61.4.15 inotify_init . . . 999
61.4.16 inotify_init1 . . . 999
61.4.17 inotify_rm_watch . . . 999
61.4.18 modify_ldt . . . 1000
61.4.19 sched_yield . . . 1000
61.4.20 setregid . . . 1000
61.4.21 setreuid . . . 1000
61.4.22 statx . . . 1000
61.4.23 sync_file_range . . . 1001
61.4.24 Sysinfo . . . 1001
61.4.25 utimensat . . . 1002
61.5 EPoll_Event . . . 1002
61.6 inotify_event . . . 1003
61.7 kernel_timespec . . . 1003
61.8 statx_timestamp . . . 1003
61.9 tstatx . . . 1003
61.10 TSysInfo . . . 1004
61.11 user_cap_data . . . 1004
61.12 user_cap_header . . . 1005
61.13 user_desc . . . 1005

62 Reference for unit 'lnfodwrf' 1006
62.1 Used units . . . 1006
62.2 Overview . . . 1006
62.3 Constants, types and variables . . . 1006
62.3.1 Types . . . 1006
62.3.2 Variables . . . 1006
62.4 Procedures and functions . . . 1007
62.4.1 CloseDwarf . . . 1007
62.4.2 DwarfBackTraceStr . . . 1007
62.4.3 GetLineInfo . . . 1007

63 Reference for unit 'Math' 1008
63.1 Used units . . . 1008
63.2 Overview . . . 1008
63.3 Cash flow functions. . . . 1008
63.4 Geometrical functions. . . . 1009
63.5 Statistical functions. . . . 1009
63.6 Number converting. . . . 1010
63.7 Exponential and logarithmic functions. . . . 1010
63.8 Hyperbolic functions. . . . 1010
63.9 Trigonometric functions. . . . 1011
63.10 Angle unit conversion. . . . 1011
63.11 Min/max determination. . . . 1011
63.12 Constants, types and variables . . . 1012
63.12.1 Constants . . . 1012
63.12.2 Types . . . 1013
63.13 Procedures and functions . . . 1014
63.13.1 ArcCos . . . 1014
63.13.2 ArcCosH . . . 1014
63.13.3 ArCosH . . . 1015
63.13.4 ArcSin . . . 1015
63.13.5 ArcSinH . . . 1016
63.13.6 ArcTan2 . . . 1016
63.13.7 ArcTanH . . . 1017
63.13.8 ArSinH . . . 1017
63.13.9 ArTanH . . . 1017
63.13.10 Ceil . . . 1018
63.13.11 Ceil64 . . . 1018
63.13.12 ClearExceptions . . . 1019
63.13.13 CompareValue . . . 1019
63.13.14 Cosecant . . . 1019
63.13.15 CosH . . . 1020
63.13.16 Cot . . . 1020
63.13.17 Cotan . . . 1020
63.13.18 Csc . . . 1021
63.13.19 CycleToRad . . . 1021
63.13.20 DegNormalize . . . 1022
63.13.21 DegToGrad . . . 1022
63.13.22 DegToRad . . . 1022
63.13.23 DivMod . . . 1023
63.13.24 EnsureRange . . . 1023

63.13.25 Floor . . . 1023
63.13.26 Floor64 . . . 1024
63.13.27 FMod . . . 1024
63.13.28 Frexp . . . 1025
63.13.29 FutureValue . . . 1025
63.13.30 GetExceptionMask . . . 1026
63.13.31 GetPrecisionMode . . . 1026
63.13.32 GetRoundMode . . . 1026
63.13.33 GradToDeg . . . 1026
63.13.34 GradToRad . . . 1027
63.13.35 Hypot . . . 1027
63.13.36 IfThen . . . 1028
63.13.37 InRange . . . 1028
63.13.38 InterestRate . . . 1028
63.13.39 IntPower . . . 1029
63.13.40 IsInfinite . . . 1029
63.13.41 IsNan . . . 1029
63.13.42 IsZero . . . 1030
63.13.43 Ldexp . . . 1030
63.13.44 LnXP1 . . . 1031
63.13.45 Log10 . . . 1031
63.13.46 Log2 . . . 1032
63.13.47 LogN . . . 1032
63.13.48 Max . . . 1033
63.13.49 MaxIntValue . . . 1033
63.13.50 MaxValue . . . 1034
63.13.51 Mean . . . 1035
63.13.52 MeanAndStdDev . . . 1036
63.13.53 Min . . . 1037
63.13.54 MinIntValue . . . 1038
63.13.55 MinValue . . . 1038
63.13.56 modulus(Float,Float):Float . . . 1039
63.13.57 MomentSkewKurtosis . . . 1040
63.13.58 Norm . . . 1041
63.13.59 NumberOfPeriods . . . 1041
63.13.60 Payment . . . 1042
63.13.61 PopnStdDev . . . 1042
63.13.62 PopnVariance . . . 1043
63.13.63 Power . . . 1044
63.13.64 power(Float,Float):Float . . . 1044

63.13.65 power(Int64,Int64):Int64 . . . 1045
63.13.66 PresentValue . . . 1045
63.13.67 RadToCycle . . . 1045
63.13.68 RadToDeg . . . 1046
63.13.69 RadToGrad . . . 1046
63.13.70 RandG . . . 1047
63.13.71 RandomFrom . . . 1047
63.13.72 RandomRange . . . 1048
63.13.73 RoundTo . . . 1048
63.13.74 SameValue . . . 1048
63.13.75 Sec . . . 1049
63.13.76 Secant . . . 1049
63.13.77 SetExceptionMask . . . 1049
63.13.78 SetPrecisionMode . . . 1050
63.13.79 SetRoundMode . . . 1050
63.13.80 Sign . . . 1050
63.13.81 SimpleRoundTo . . . 1050
63.13.82 SinCos . . . 1051
63.13.83 SinH . . . 1052
63.13.84 StdDev . . . 1052
63.13.85 Sum . . . 1053
63.13.86 SumInt . . . 1054
63.13.87 SumOfSquares . . . 1054
63.13.88 SumsAndSquares . . . 1055
63.13.89 Tan . . . 1056
63.13.90 TanH . . . 1056
63.13.91 TotalVariance . . . 1057
63.13.92 Variance . . . 1058
63.14 EInvalidArgument . . . 1059
63.14.1 Description . . . 1059

64 Reference for unit 'matrix' 1060
64.1 Used units . . . 1060
64.2 Overview . . . 1060
64.3 Constants, types and variables . . . 1061
64.3.1 Types . . . 1061
64.4 Procedures and functions . . . 1063
64.4.1 add(Tmatrix2_double,Double):Tmatrix2_double . . . 1063
64.4.2 add(Tmatrix2_double,Tmatrix2_double):Tmatrix2_double . . . 1063
64.4.3 add(Tmatrix2_extended,extended):Tmatrix2_extended . . . 1064

64.4.4 add(Tmatrix2_extended,Tmatrix2_extended):Tmatrix2_extended 1064
64.4.5 add(Tmatrix2_single,single):Tmatrix2_single 1064
64.4.6 add(Tmatrix2_single,Tmatrix2_single):Tmatrix2_single 1064
64.4.7 add(Tmatrix3_double,Double):Tmatrix3_double 1064
64.4.8 add(Tmatrix3_double,Tmatrix3_double):Tmatrix3_double 1065
64.4.9 add(Tmatrix3_extended,extended):Tmatrix3_extended 1065
64.4.10 add(Tmatrix3_extended,Tmatrix3_extended):Tmatrix3_extended 1065
64.4.11 add(Tmatrix3_single,single):Tmatrix3_single 1065
64.4.12 add(Tmatrix3_single,Tmatrix3_single):Tmatrix3_single 1065
64.4.13 add(Tmatrix4_double,Double):Tmatrix4_double 1066
64.4.14 add(Tmatrix4_double,Tmatrix4_double):Tmatrix4_double 1066
64.4.15 add(Tmatrix4_extended,extended):Tmatrix4_extended 1066
64.4.16 add(Tmatrix4_extended,Tmatrix4_extended):Tmatrix4_extended 1066
64.4.17 add(Tmatrix4_single,single):Tmatrix4_single 1066
64.4.18 add(Tmatrix4_single,Tmatrix4_single):Tmatrix4_single 1067
64.4.19 add(Tvector2_double,Double):Tvector2_double 1067
64.4.20 add(Tvector2_double,Tvector2_double):Tvector2_double 1067
64.4.21 add(Tvector2_extended,extended):Tvector2_extended 1067
64.4.22 add(Tvector2_extended,Tvector2_extended):Tvector2_extended 1067
64.4.23 add(Tvector2_single,single):Tvector2_single 1068
64.4.24 add(Tvector2_single,Tvector2_single):Tvector2_single 1068
64.4.25 add(Tvector3_double,Double):Tvector3_double 1068
64.4.26 add(Tvector3_double,Tvector3_double):Tvector3_double 1068
64.4.27 add(Tvector3_extended,extended):Tvector3_extended 1068
64.4.28 add(Tvector3_extended,Tvector3_extended):Tvector3_extended 1069
64.4.29 add(Tvector3_single,single):Tvector3_single 1069
64.4.30 add(Tvector3_single,Tvector3_single):Tvector3_single 1069
64.4.31 add(Tvector4_double,Double):Tvector4_double 1069
64.4.32 add(Tvector4_double,Tvector4_double):Tvector4_double 1069
64.4.33 add(Tvector4_extended,extended):Tvector4_extended 1070
64.4.34 add(Tvector4_extended,Tvector4_extended):Tvector4_extended 1070
64.4.35 add(Tvector4_single,single):Tvector4_single 1070
64.4.36 add(Tvector4_single,Tvector4_single):Tvector4_single 1070
64.4.37 assign(Tmatrix2_double):Tmatrix2_extended 1070
64.4.38 assign(Tmatrix2_double):Tmatrix2_single 1071
64.4.39 assign(Tmatrix2_double):Tmatrix3_double 1071
64.4.40 assign(Tmatrix2_double):Tmatrix3_extended 1071
64.4.41 assign(Tmatrix2_double):Tmatrix3_single 1071
64.4.42 assign(Tmatrix2_double):Tmatrix4_double 1071
64.4.43 assign(Tmatrix2_double):Tmatrix4_extended 1072

64.4.44 assign(Tmatrix2_double):Tmatrix4_single . . . 1072
64.4.45 assign(Tmatrix2_extended):Tmatrix2_double . . . 1072
64.4.46 assign(Tmatrix2_extended):Tmatrix2_single . . . 1072
64.4.47 assign(Tmatrix2_extended):Tmatrix3_double . . . 1073
64.4.48 assign(Tmatrix2_extended):Tmatrix3_extended . . . 1073
64.4.49 assign(Tmatrix2_extended):Tmatrix3_single . . . 1073
64.4.50 assign(Tmatrix2_extended):Tmatrix4_double . . . 1073
64.4.51 assign(Tmatrix2_extended):Tmatrix4_extended . . . 1074
64.4.52 assign(Tmatrix2_extended):Tmatrix4_single . . . 1074
64.4.53 assign(Tmatrix2_single):Tmatrix2_double . . . 1074
64.4.54 assign(Tmatrix2_single):Tmatrix2_extended . . . 1074
64.4.55 assign(Tmatrix2_single):Tmatrix3_double . . . 1074
64.4.56 assign(Tmatrix2_single):Tmatrix3_extended . . . 1075
64.4.57 assign(Tmatrix2_single):Tmatrix3_single . . . 1075
64.4.58 assign(Tmatrix2_single):Tmatrix4_double . . . 1075
64.4.59 assign(Tmatrix2_single):Tmatrix4_extended . . . 1075
64.4.60 assign(Tmatrix2_single):Tmatrix4_single . . . 1075
64.4.61 assign(Tmatrix3_double):Tmatrix2_double . . . 1076
64.4.62 assign(Tmatrix3_double):Tmatrix2_extended . . . 1076
64.4.63 assign(Tmatrix3_double):Tmatrix2_single . . . 1076
64.4.64 assign(Tmatrix3_double):Tmatrix3_extended . . . 1076
64.4.65 assign(Tmatrix3_double):Tmatrix3_single . . . 1076
64.4.66 assign(Tmatrix3_double):Tmatrix4_double . . . 1077
64.4.67 assign(Tmatrix3_double):Tmatrix4_extended . . . 1077
64.4.68 assign(Tmatrix3_double):Tmatrix4_single . . . 1077
64.4.69 assign(Tmatrix3_extended):Tmatrix2_double . . . 1077
64.4.70 assign(Tmatrix3_extended):Tmatrix2_extended . . . 1078
64.4.71 assign(Tmatrix3_extended):Tmatrix2_single . . . 1078
64.4.72 assign(Tmatrix3_extended):Tmatrix3_double . . . 1078
64.4.73 assign(Tmatrix3_extended):Tmatrix3_single . . . 1078
64.4.74 assign(Tmatrix3_extended):Tmatrix4_double . . . 1079
64.4.75 assign(Tmatrix3_extended):Tmatrix4_extended . . . 1079
64.4.76 assign(Tmatrix3_extended):Tmatrix4_single . . . 1079
64.4.77 assign(Tmatrix3_single):Tmatrix2_double . . . 1079
64.4.78 assign(Tmatrix3_single):Tmatrix2_extended . . . 1079
64.4.79 assign(Tmatrix3_single):Tmatrix2_single . . . 1080
64.4.80 assign(Tmatrix3_single):Tmatrix3_double . . . 1080
64.4.81 assign(Tmatrix3_single):Tmatrix3_extended . . . 1080
64.4.82 assign(Tmatrix3_single):Tmatrix4_double . . . 1080
64.4.83 assign(Tmatrix3_single):Tmatrix4_extended . . . 1081

64.4.84 assign(Tmatrix3_single):Tmatrix4_single . . . 1081
64.4.85 assign(Tmatrix4_double):Tmatrix2_double . . . 1081
64.4.86 assign(Tmatrix4_double):Tmatrix2_extended . . . 1081
64.4.87 assign(Tmatrix4_double):Tmatrix2_single . . . 1081
64.4.88 assign(Tmatrix4_double):Tmatrix3_double . . . 1082
64.4.89 assign(Tmatrix4_double):Tmatrix3_extended . . . 1082
64.4.90 assign(Tmatrix4_double):Tmatrix3_single . . . 1082
64.4.91 assign(Tmatrix4_double):Tmatrix4_extended . . . 1082
64.4.92 assign(Tmatrix4_double):Tmatrix4_single . . . 1082
64.4.93 assign(Tmatrix4_extended):Tmatrix2_double . . . 1083
64.4.94 assign(Tmatrix4_extended):Tmatrix2_extended . . . 1083
64.4.95 assign(Tmatrix4_extended):Tmatrix2_single . . . 1083
64.4.96 assign(Tmatrix4_extended):Tmatrix3_double . . . 1083
64.4.97 assign(Tmatrix4_extended):Tmatrix3_extended . . . 1084
64.4.98 assign(Tmatrix4_extended):Tmatrix3_single . . . 1084
64.4.99 assign(Tmatrix4_extended):Tmatrix4_double . . . 1084
64.4.100 assign(Tmatrix4_extended):Tmatrix4_single . . . 1084
64.4.101 assign(Tmatrix4_single):Tmatrix2_double . . . 1084
64.4.102 assign(Tmatrix4_single):Tmatrix2_extended . . . 1085
64.4.103 assign(Tmatrix4_single):Tmatrix2_single . . . 1085
64.4.104 assign(Tmatrix4_single):Tmatrix3_double . . . 1085
64.4.105 assign(Tmatrix4_single):Tmatrix3_extended . . . 1085
64.4.106 assign(Tmatrix4_single):Tmatrix3_single . . . 1086
64.4.107 assign(Tmatrix4_single):Tmatrix4_double . . . 1086
64.4.108 assign(Tmatrix4_single):Tmatrix4_extended . . . 1086
64.4.109 assign(Tvector2_double):Tvector2_extended . . . 1086
64.4.110 assign(Tvector2_double):Tvector2_single . . . 1086
64.4.111 assign(Tvector2_double):Tvector3_double . . . 1087
64.4.112 assign(Tvector2_double):Tvector3_extended . . . 1087
64.4.113 assign(Tvector2_double):Tvector3_single . . . 1087
64.4.114 assign(Tvector2_double):Tvector4_double . . . 1087
64.4.115 assign(Tvector2_double):Tvector4_extended . . . 1087
64.4.116 assign(Tvector2_double):Tvector4_single . . . 1088
64.4.117 assign(Tvector2_extended):Tvector2_double . . . 1088
64.4.118 assign(Tvector2_extended):Tvector2_single . . . 1088
64.4.119 assign(Tvector2_extended):Tvector3_double . . . 1088
64.4.120 assign(Tvector2_extended):Tvector3_extended . . . 1089
64.4.121 assign(Tvector2_extended):Tvector3_single . . . 1089
64.4.122 assign(Tvector2_extended):Tvector4_double . . . 1089
64.4.123 assign(Tvector2_extended):Tvector4_extended . . . 1089

64.4.124 assign(Tvector2_extended):Tvector4_single . . . 1090
64.4.125 assign(Tvector2_single):Tvector2_double . . . 1090
64.4.126 assign(Tvector2_single):Tvector2_extended . . . 1090
64.4.127 assign(Tvector2_single):Tvector3_double . . . 1090
64.4.128 assign(Tvector2_single):Tvector3_extended . . . 1090
64.4.129 assign(Tvector2_single):Tvector3_single . . . 1091
64.4.130 assign(Tvector2_single):Tvector4_double . . . 1091
64.4.131 assign(Tvector2_single):Tvector4_extended . . . 1091
64.4.132 assign(Tvector2_single):Tvector4_single . . . 1091
64.4.133 assign(Tvector3_double):Tvector2_double . . . 1092
64.4.134 assign(Tvector3_double):Tvector2_extended . . . 1092
64.4.135 assign(Tvector3_double):Tvector2_single . . . 1092
64.4.136 assign(Tvector3_double):Tvector3_extended . . . 1092
64.4.137 assign(Tvector3_double):Tvector3_single . . . 1092
64.4.138 assign(Tvector3_double):Tvector4_double . . . 1093
64.4.139 assign(Tvector3_double):Tvector4_extended . . . 1093
64.4.140 assign(Tvector3_double):Tvector4_single . . . 1093
64.4.141 assign(Tvector3_extended):Tvector2_double . . . 1093
64.4.142 assign(Tvector3_extended):Tvector2_extended . . . 1094
64.4.143 assign(Tvector3_extended):Tvector2_single . . . 1094
64.4.144 assign(Tvector3_extended):Tvector3_double . . . 1094
64.4.145 assign(Tvector3_extended):Tvector3_single . . . 1094
64.4.146 assign(Tvector3_extended):Tvector4_double . . . 1095
64.4.147 assign(Tvector3_extended):Tvector4_extended . . . 1095
64.4.148 assign(Tvector3_extended):Tvector4_single . . . 1095
64.4.149 assign(Tvector3_single):Tvector2_double . . . 1095
64.4.150 assign(Tvector3_single):Tvector2_extended . . . 1095
64.4.151 assign(Tvector3_single):Tvector2_single . . . 1096
64.4.152 assign(Tvector3_single):Tvector3_double . . . 1096
64.4.153 assign(Tvector3_single):Tvector3_extended . . . 1096
64.4.154 assign(Tvector3_single):Tvector4_double . . . 1096
64.4.155 assign(Tvector3_single):Tvector4_extended . . . 1097
64.4.156 assign(Tvector3_single):Tvector4_single . . . 1097
64.4.157 assign(Tvector4_double):Tvector2_double . . . 1097
64.4.158 assign(Tvector4_double):Tvector2_extended . . . 1097
64.4.159 assign(Tvector4_double):Tvector2_single . . . 1097
64.4.160 assign(Tvector4_double):Tvector3_double . . . 1098
64.4.161 assign(Tvector4_double):Tvector3_extended . . . 1098
64.4.162 assign(Tvector4_double):Tvector3_single . . . 1098
64.4.163 assign(Tvector4_double):Tvector4_extended . . . 1098

64.4.164 assign(Tvector4_double):Tvector4_single . . . 1099
64.4.165 assign(Tvector4_extended):Tvector2_double . . . 1099
64.4.166 assign(Tvector4_extended):Tvector2_extended . . . 1099
64.4.167 assign(Tvector4_extended):Tvector2_single . . . 1099
64.4.168 assign(Tvector4_extended):Tvector3_double . . . 1100
64.4.169 assign(Tvector4_extended):Tvector3_extended . . . 1100
64.4.170 assign(Tvector4_extended):Tvector3_single . . . 1100
64.4.171 assign(Tvector4_extended):Tvector4_double . . . 1100
64.4.172 assign(Tvector4_extended):Tvector4_single . . . 1101
64.4.173 assign(Tvector4_single):Tvector2_double . . . 1101
64.4.174 assign(Tvector4_single):Tvector2_extended . . . 1101
64.4.175 assign(Tvector4_single):Tvector2_single . . . 1101
64.4.176 assign(Tvector4_single):Tvector3_double . . . 1102
64.4.177 assign(Tvector4_single):Tvector3_extended . . . 1102
64.4.178 assign(Tvector4_single):Tvector3_single . . . 1102
64.4.179 assign(Tvector4_single):Tvector4_double . . . 1102
64.4.180 assign(Tvector4_single):Tvector4_extended . . . 1102
64.4.181 divide(Tmatrix2_double,Double):Tmatrix2_double . . . 1103
64.4.182 divide(Tmatrix2_extended,extended):Tmatrix2_extended . . . 1103
64.4.183 divide(Tmatrix2_single,single):Tmatrix2_single . . . 1103
64.4.184 divide(Tmatrix3_double,Double):Tmatrix3_double . . . 1103
64.4.185 divide(Tmatrix3_extended,extended):Tmatrix3_extended . . . 1103
64.4.186 divide(Tmatrix3_single,single):Tmatrix3_single . . . 1104
64.4.187 divide(Tmatrix4_double,Double):Tmatrix4_double . . . 1104
64.4.188 divide(Tmatrix4_extended,extended):Tmatrix4_extended . . . 1104
64.4.189 divide(Tmatrix4_single,single):Tmatrix4_single . . . 1104
64.4.190 divide(Tvector2_double,Double):Tvector2_double . . . 1104
64.4.191 divide(Tvector2_extended,extended):Tvector2_extended . . . 1105
64.4.192 divide(Tvector2_single,single):Tvector2_single . . . 1105
64.4.193 divide(Tvector3_double,Double):Tvector3_double . . . 1105
64.4.194 divide(Tvector3_extended,extended):Tvector3_extended . . . 1105
64.4.195 divide(Tvector3_single,single):Tvector3_single . . . 1105
64.4.196 divide(Tvector4_double,Double):Tvector4_double . . . 1106
64.4.197 divide(Tvector4_extended,extended):Tvector4_extended . . . 1106
64.4.198 divide(Tvector4_single,single):Tvector4_single . . . 1106
64.4.199 multiply(Tmatrix2_double,Double):Tmatrix2_double . . . 1106
64.4.200 multiply(Tmatrix2_double,Tmatrix2_double):Tmatrix2_double . . . 1106
64.4.201 multiply(Tmatrix2_double,Tvector2_double):Tvector2_double . . . 1107
64.4.202 multiply(Tmatrix2_extended,extended):Tmatrix2_extended . . . 1107
64.4.203 multiply(Tmatrix2_extended,Tmatrix2_extended):Tmatrix2_extended . . . 1107

64.4.204 multiply(Tmatrix2_extended,Tvector2_extended):Tvector2_extended . . . 1107
64.4.205 multiply(Tmatrix2_single,single):Tmatrix2_single 1108
64.4.206 multiply(Tmatrix2_single,Tmatrix2_single):Tmatrix2_single 1108
64.4.207 multiply(Tmatrix2_single,Tvector2_single):Tvector2_single 1108
64.4.208 multiply(Tmatrix3_double,Double):Tmatrix3_double 1108
64.4.209 multiply(Tmatrix3_double,Tmatrix3_double):Tmatrix3_double 1108
64.4.210 multiply(Tmatrix3_double,Tvector3_double):Tvector3_double 1109
64.4.211 multiply(Tmatrix3_extended,extended):Tmatrix3_extended 1109
64.4.212 multiply(Tmatrix3_extended,Tmatrix3_extended):Tmatrix3_extended . . 1109
64.4.213 multiply(Tmatrix3_extended,Tvector3_extended):Tvector3_extended . . . 1109
64.4.214 multiply(Tmatrix3_single,single):Tmatrix3_single 1110
64.4.215 multiply(Tmatrix3_single,Tmatrix3_single):Tmatrix3_single 1110
64.4.216 multiply(Tmatrix3_single,Tvector3_single):Tvector3_single 1110
64.4.217 multiply(Tmatrix4_double,Double):Tmatrix4_double 1110
64.4.218 multiply(Tmatrix4_double,Tmatrix4_double):Tmatrix4_double 1110
64.4.219 multiply(Tmatrix4_double,Tvector4_double):Tvector4_double 1111
64.4.220 multiply(Tmatrix4_extended,extended):Tmatrix4_extended 1111
64.4.221 multiply(Tmatrix4_extended,Tmatrix4_extended):Tmatrix4_extended . . 1111
64.4.222 multiply(Tmatrix4_extended,Tvector4_extended):Tvector4_extended . . . 1111
64.4.223 multiply(Tmatrix4_single,single):Tmatrix4_single 1112
64.4.224 multiply(Tmatrix4_single,Tmatrix4_single):Tmatrix4_single 1112
64.4.225 multiply(Tmatrix4_single,Tvector4_single):Tvector4_single 1112
64.4.226 multiply(Tvector2_double,Double):Tvector2_double 1112
64.4.227 multiply(Tvector2_double,Tvector2_double):Tvector2_double 1112
64.4.228 multiply(Tvector2_extended,extended):Tvector2_extended 1113
64.4.229 multiply(Tvector2_extended,Tvector2_extended):Tvector2_extended . . . 1113
64.4.230 multiply(Tvector2_single,single):Tvector2_single 1113
64.4.231 multiply(Tvector2_single,Tvector2_single):Tvector2_single 1113
64.4.232 multiply(Tvector3_double,Double):Tvector3_double 1113
64.4.233 multiply(Tvector3_double,Tvector3_double):Tvector3_double 1114
64.4.234 multiply(Tvector3_extended,extended):Tvector3_extended 1114
64.4.235 multiply(Tvector3_extended,Tvector3_extended):Tvector3_extended . . . 1114
64.4.236 multiply(Tvector3_single,single):Tvector3_single 1114
64.4.237 multiply(Tvector3_single,Tvector3_single):Tvector3_single 1114
64.4.238 multiply(Tvector4_double,Double):Tvector4_double 1115
64.4.239 multiply(Tvector4_double,Tvector4_double):Tvector4_double 1115
64.4.240 multiply(Tvector4_extended,extended):Tvector4_extended 1115
64.4.241 multiply(Tvector4_extended,Tvector4_extended):Tvector4_extended . . . 1115
64.4.242 multiply(Tvector4_single,single):Tvector4_single 1115
64.4.243 multiply(Tvector4_single,Tvector4_single):Tvector4_single 1116

64.4.244 negative(Tmatrix2_double):Tmatrix2_double 1116
64.4.245 negative(Tmatrix2_extended):Tmatrix2_extended 1116
64.4.246 negative(Tmatrix2_single):Tmatrix2_single 1116
64.4.247 negative(Tmatrix3_double):Tmatrix3_double 1116
64.4.248 negative(Tmatrix3_extended):Tmatrix3_extended 1116
64.4.249 negative(Tmatrix3_single):Tmatrix3_single 1117
64.4.250 negative(Tmatrix4_double):Tmatrix4_double 1117
64.4.251 negative(Tmatrix4_extended):Tmatrix4_extended 1117
64.4.252 negative(Tmatrix4_single):Tmatrix4_single 1117
64.4.253 negative(Tvector2_double):Tvector2_double 1117
64.4.254 negative(Tvector2_extended):Tvector2_extended 1117
64.4.255 negative(Tvector2_single):Tvector2_single 1118
64.4.256 negative(Tvector3_double):Tvector3_double 1118
64.4.257 negative(Tvector3_extended):Tvector3_extended 1118
64.4.258 negative(Tvector3_single):Tvector3_single 1118
64.4.259 negative(Tvector4_double):Tvector4_double 1118
64.4.260 negative(Tvector4_extended):Tvector4_extended 1119
64.4.261 negative(Tvector4_single):Tvector4_single 1119
64.4.262 power(Tvector2_double,Tvector2_double):Double 1119
64.4.263 power(Tvector2_extended,Tvector2_extended):extended 1119
64.4.264 power(Tvector2_single,Tvector2_single):single 1119
64.4.265 power(Tvector3_double,Tvector3_double):Double 1120
64.4.266 power(Tvector3_extended,Tvector3_extended):extended 1120
64.4.267 power(Tvector3_single,Tvector3_single):single 1120
64.4.268 power(Tvector4_double,Tvector4_double):Double 1120
64.4.269 power(Tvector4_extended,Tvector4_extended):extended 1120
64.4.270 power(Tvector4_single,Tvector4_single):single 1121
64.4.271 subtract(Tmatrix2_double,Double):Tmatrix2_double 1121
64.4.272 subtract(Tmatrix2_double,Tmatrix2_double):Tmatrix2_double 1121
64.4.273 subtract(Tmatrix2_extended,extended):Tmatrix2_extended 1121
64.4.274 subtract(Tmatrix2_extended,Tmatrix2_extended):Tmatrix2_extended . . 1121
64.4.275 subtract(Tmatrix2_single,single):Tmatrix2_single 1122
64.4.276 subtract(Tmatrix2_single,Tmatrix2_single):Tmatrix2_single 1122
64.4.277 subtract(Tmatrix3_double,Double):Tmatrix3_double 1122
64.4.278 subtract(Tmatrix3_double,Tmatrix3_double):Tmatrix3_double 1122
64.4.279 subtract(Tmatrix3_extended,extended):Tmatrix3_extended 1122
64.4.280 subtract(Tmatrix3_extended,Tmatrix3_extended):Tmatrix3_extended . . 1123
64.4.281 subtract(Tmatrix3_single,single):Tmatrix3_single 1123
64.4.282 subtract(Tmatrix3_single,Tmatrix3_single):Tmatrix3_single 1123
64.4.283 subtract(Tmatrix4_double,Double):Tmatrix4_double 1123

64.4.284 subtract(Tmatrix4_double,Tmatrix4_double):Tmatrix4_double 1123
64.4.285 subtract(Tmatrix4_extended,extended):Tmatrix4_extended 1124
64.4.286 subtract(Tmatrix4_extended,Tmatrix4_extended):Tmatrix4_extended . . 1124
64.4.287 subtract(Tmatrix4_single,single):Tmatrix4_single 1124
64.4.288 subtract(Tmatrix4_single,Tmatrix4_single):Tmatrix4_single 1124
64.4.289 subtract(Tvector2_double,Double):Tvector2_double 1124
64.4.290 subtract(Tvector2_double,Tvector2_double):Tvector2_double 1125
64.4.291 subtract(Tvector2_extended,extended):Tvector2_extended 1125
64.4.292 subtract(Tvector2_extended,Tvector2_extended):Tvector2_extended . . . 1125
64.4.293 subtract(Tvector2_single,single):Tvector2_single 1125
64.4.294 subtract(Tvector2_single,Tvector2_single):Tvector2_single 1125
64.4.295 subtract(Tvector3_double,Double):Tvector3_double 1126
64.4.296 subtract(Tvector3_double,Tvector3_double):Tvector3_double 1126
64.4.297 subtract(Tvector3_extended,extended):Tvector3_extended 1126
64.4.298 subtract(Tvector3_extended,Tvector3_extended):Tvector3_extended . . . 1126
64.4.299 subtract(Tvector3_single,single):Tvector3_single 1126
64.4.300 subtract(Tvector3_single,Tvector3_single):Tvector3_single 1127
64.4.301 subtract(Tvector4_double,Double):Tvector4_double 1127
64.4.302 subtract(Tvector4_double,Tvector4_double):Tvector4_double 1127
64.4.303 subtract(Tvector4_extended,extended):Tvector4_extended 1127
64.4.304 subtract(Tvector4_extended,Tvector4_extended):Tvector4_extended . . . 1127
64.4.305 subtract(Tvector4_single,single):Tvector4_single 1128
64.4.306 subtract(Tvector4_single,Tvector4_single):Tvector4_single 1128
64.4.307 symmetricaldifference(Tvector3_double,Tvector3_double):Tvector3_double1128
64.4.308 symmetricaldifference(Tvector3_extended,Tvector3_extended):Tvector3_extended1128
64.4.309 symmetricaldifference(Tvector3_single,Tvector3_single):Tvector3_single 1129
64.5 Tmatrix2_double 1129
64.5.1 Description 1129
64.5.2 Method overview 1129
64.5.3 Tmatrix2_double.init_zero 1129
64.5.4 Tmatrix2_double.init_identity 1129
64.5.5 Tmatrix2_double.init 1130
64.5.6 Tmatrix2_double.get_column 1130
64.5.7 Tmatrix2_double.get_row 1130
64.5.8 Tmatrix2_double.set_column 1130
64.5.9 Tmatrix2_double.set_row 1130
64.5.10 Tmatrix2_double.determinant 1130
64.5.11 Tmatrix2_double.inverse 1131
64.5.12 Tmatrix2_double.transpose 1131
64.6 Tmatrix2_extended 1131

64.6.1 Description . . . 1131
64.6.2 Method overview . . . 1131
64.6.3 Tmatrix2_extended.init_zero . . . 1131
64.6.4 Tmatrix2_extended.init_identity . . . 1132
64.6.5 Tmatrix2_extended.init . . . 1132
64.6.6 Tmatrix2_extended.get_column . . . 1132
64.6.7 Tmatrix2_extended.get_row . . . 1132
64.6.8 Tmatrix2_extended.set_column . . . 1132
64.6.9 Tmatrix2_extended.set_row . . . 1132
64.6.10 Tmatrix2_extended.determinant . . . 1133
64.6.11 Tmatrix2_extended.inverse . . . 1133
64.6.12 Tmatrix2_extended.transpose . . . 1133
64.7 Tmatrix2_single . . . 1133
64.7.1 Description . . . 1133
64.7.2 Method overview . . . 1133
64.7.3 Tmatrix2_single.init_zero . . . 1134
64.7.4 Tmatrix2_single.init_identity . . . 1134
64.7.5 Tmatrix2_single.init . . . 1134
64.7.6 Tmatrix2_single.get_column . . . 1134
64.7.7 Tmatrix2_single.get_row . . . 1134
64.7.8 Tmatrix2_single.set_column . . . 1134
64.7.9 Tmatrix2_single.set_row . . . 1135
64.7.10 Tmatrix2_single.determinant . . . 1135
64.7.11 Tmatrix2_single.inverse . . . 1135
64.7.12 Tmatrix2_single.transpose . . . 1135
64.8 Tmatrix3_double . . . 1135
64.8.1 Description . . . 1135
64.8.2 Method overview . . . 1136
64.8.3 Tmatrix3_double.init_zero . . . 1136
64.8.4 Tmatrix3_double.init_identity . . . 1136
64.8.5 Tmatrix3_double.init . . . 1136
64.8.6 Tmatrix3_double.get_column . . . 1136
64.8.7 Tmatrix3_double.get_row . . . 1137
64.8.8 Tmatrix3_double.set_column . . . 1137
64.8.9 Tmatrix3_double.set_row . . . 1137
64.8.10 Tmatrix3_double.determinant . . . 1137
64.8.11 Tmatrix3_double.inverse . . . 1137
64.8.12 Tmatrix3_double.transpose . . . 1137
64.9 Tmatrix3_extended . . . 1138
64.9.1 Description . . . 1138

64.9.2 Method overview . . . 1138
64.9.3 Tmatrix3_extended.init_zero . . . 1138
64.9.4 Tmatrix3_extended.init_identity . . . 1138
64.9.5 Tmatrix3_extended.init . . . 1138
64.9.6 Tmatrix3_extended.get_column . . . 1139
64.9.7 Tmatrix3_extended.get_row . . . 1139
64.9.8 Tmatrix3_extended.set_column . . . 1139
64.9.9 Tmatrix3_extended.set_row . . . 1139
64.9.10 Tmatrix3_extended.determinant . . . 1139
64.9.11 Tmatrix3_extended.inverse . . . 1139
64.9.12 Tmatrix3_extended.transpose . . . 1140
64.10 Tmatrix3_single . . . 1140
64.10.1 Description . . . 1140
64.10.2 Method overview . . . 1140
64.10.3 Tmatrix3_single.init_zero . . . 1140
64.10.4 Tmatrix3_single.init_identity . . . 1140
64.10.5 Tmatrix3_single.init . . . 1141
64.10.6 Tmatrix3_single.get_column . . . 1141
64.10.7 Tmatrix3_single.get_row . . . 1141
64.10.8 Tmatrix3_single.set_column . . . 1141
64.10.9 Tmatrix3_single.set_row . . . 1141
64.10.10 Tmatrix3_single.determinant . . . 1142
64.10.11 Tmatrix3_single.inverse . . . 1142
64.10.12 Tmatrix3_single.transpose . . . 1142
64.11 Tmatrix4_double . . . 1142
64.11.1 Description . . . 1142
64.11.2 Method overview . . . 1142
64.11.3 Tmatrix4_double.init_zero . . . 1143
64.11.4 Tmatrix4_double.init_identity . . . 1143
64.11.5 Tmatrix4_double.init . . . 1143
64.11.6 Tmatrix4_double.get_column . . . 1143
64.11.7 Tmatrix4_double.get_row . . . 1143
64.11.8 Tmatrix4_double.set_column . . . 1144
64.11.9 Tmatrix4_double.set_row . . . 1144
64.11.10 Tmatrix4_double.determinant . . . 1144
64.11.11 Tmatrix4_double.inverse . . . 1144
64.11.12 Tmatrix4_double.transpose . . . 1144
64.12 Tmatrix4_extended . . . 1145
64.12.1 Description . . . 1145
64.12.2 Method overview . . . 1145

64.12.3 Tmatrix4_extended.init_zero . . . 1145
64.12.4 Tmatrix4_extended.init_identity . . . 1145
64.12.5 Tmatrix4_extended.init . . . 1145
64.12.6 Tmatrix4_extended.get_column . . . 1146
64.12.7 Tmatrix4_extended.get_row . . . 1146
64.12.8 Tmatrix4_extended.set_column . . . 1146
64.12.9 Tmatrix4_extended.set_row . . . 1146
64.12.10 Tmatrix4_extended.determinant . . . 1146
64.12.11 Tmatrix4_extended.inverse . . . 1147
64.12.12 Tmatrix4_extended.transpose . . . 1147
64.13 Tmatrix4_single . . . 1147
64.13.1 Description . . . 1147
64.13.2 Method overview . . . 1147
64.13.3 Tmatrix4_single.init_zero . . . 1147
64.13.4 Tmatrix4_single.init_identity . . . 1148
64.13.5 Tmatrix4_single.init . . . 1148
64.13.6 Tmatrix4_single.get_column . . . 1148
64.13.7 Tmatrix4_single.get_row . . . 1148
64.13.8 Tmatrix4_single.set_column . . . 1148
64.13.9 Tmatrix4_single.set_row . . . 1149
64.13.10 Tmatrix4_single.determinant . . . 1149
64.13.11 Tmatrix4_single.inverse . . . 1149
64.13.12 Tmatrix4_single.transpose . . . 1149
64.14 Tvector2_double . . . 1149
64.14.1 Description . . . 1149
64.14.2 Method overview . . . 1150
64.14.3 Tvector2_double.init_zero . . . 1150
64.14.4 Tvector2_double.init_one . . . 1150
64.14.5 Tvector2_double.init . . . 1150
64.14.6 Tvector2_double.length . . . 1150
64.14.7 Tvector2_double.squared_length . . . 1150
64.15 Tvector2_extended . . . 1151
64.15.1 Description . . . 1151
64.15.2 Method overview . . . 1151
64.15.3 Tvector2_extended.init_zero . . . 1151
64.15.4 Tvector2_extended.init_one . . . 1151
64.15.5 Tvector2_extended.init . . . 1151
64.15.6 Tvector2_extended.length . . . 1151
64.15.7 Tvector2_extended.squared_length . . . 1152
64.16 Tvector2_single . . . 1152

64.16.1 Description . . . 1152
64.16.2 Method overview . . . 1152
64.16.3 Tvector2_single.init_zero . . . 1152
64.16.4 Tvector2_single.init_one . . . 1152
64.16.5 Tvector2_single.init . . . 1152
64.16.6 Tvector2_single.length . . . 1153
64.16.7 Tvector2_single.squared_length . . . 1153
64.17 Tvector3_double . . . 1153
64.17.1 Description . . . 1153
64.17.2 Method overview . . . 1153
64.17.3 Tvector3_double.init_zero . . . 1153
64.17.4 Tvector3_double.init_one . . . 1153
64.17.5 Tvector3_double.init . . . 1154
64.17.6 Tvector3_double.length . . . 1154
64.17.7 Tvector3_double.squared_length . . . 1154
64.18 Tvector3_extended . . . 1154
64.18.1 Description . . . 1154
64.18.2 Method overview . . . 1154
64.18.3 Tvector3_extended.init_zero . . . 1154
64.18.4 Tvector3_extended.init_one . . . 1155
64.18.5 Tvector3_extended.init . . . 1155
64.18.6 Tvector3_extended.length . . . 1155
64.18.7 Tvector3_extended.squared_length . . . 1155
64.19 Tvector3_single . . . 1155
64.19.1 Description . . . 1155
64.19.2 Method overview . . . 1155
64.19.3 Tvector3_single.init_zero . . . 1156
64.19.4 Tvector3_single.init_one . . . 1156
64.19.5 Tvector3_single.init . . . 1156
64.19.6 Tvector3_single.length . . . 1156
64.19.7 Tvector3_single.squared_length . . . 1156
64.20 Tvector4_double . . . 1156
64.20.1 Description . . . 1156
64.20.2 Method overview . . . 1157
64.20.3 Tvector4_double.init_zero . . . 1157
64.20.4 Tvector4_double.init_one . . . 1157
64.20.5 Tvector4_double.init . . . 1157
64.20.6 Tvector4_double.length . . . 1157
64.20.7 Tvector4_double.squared_length . . . 1157
64.21 Tvector4_extended . . . 1158

64.21.1 Description . . . 1158
64.21.2 Method overview . . . 1158
64.21.3 Tvector4_extended.init_zero . . . 1158
64.21.4 Tvector4_extended.init_one . . . 1158
64.21.5 Tvector4_extended.init . . . 1158
64.21.6 Tvector4_extended.length . . . 1158
64.21.7 Tvector4_extended.squared_length . . . 1159
64.22 Tvector4_single . . . 1159
64.22.1 Description . . . 1159
64.22.2 Method overview . . . 1159
64.22.3 Tvector4_single.init_zero . . . 1159
64.22.4 Tvector4_single.init_one . . . 1159
64.22.5 Tvector4_single.init . . . 1159
64.22.6 Tvector4_single.length . . . 1160
64.22.7 Tvector4_single.squared_length . . . 1160

65 Reference for unit 'mmx' 1161
65.1 Used units . . . 1161
65.2 Overview . . . 1161
65.3 Constants, types and variables . . . 1161
65.3.1 Constants . . . 1161
65.3.2 Types . . . 1162
65.4 Procedures and functions . . . 1163
65.4.1 emms . . . 1163
65.4.2 femms . . . 1164

66 Reference for unit 'Mouse' 1165
66.1 Used units . . . 1165
66.2 Overview . . . 1165
66.3 Writing a custom mouse driver. . . . 1165
66.4 Constants, types and variables . . . 1167
66.4.1 Constants . . . 1167
66.4.2 Types . . . 1168
66.4.3 Variables . . . 1169
66.5 Procedures and functions . . . 1169
66.5.1 DetectMouse . . . 1169
66.5.2 DoneMouse . . . 1170
66.5.3 GetMouseButtons . . . 1170
66.5.4 GetMouseDriver . . . 1171
66.5.5 GetMouseEvent . . . 1171
66.5.6 GetMouseX . . . 1171

66.5.7 GetMouseY . . . 1172
66.5.8 HideMouse . . . 1172
66.5.9 InitMouse . . . 1173
66.5.10 PollMouseEvent . . . 1173
66.5.11 PutMouseEvent . . . 1174
66.5.12 SetMouseDriver . . . 1174
66.5.13 SetMouseXY . . . 1174
66.5.14 ShowMouse . . . 1175
66.6 TMouseDriver . . . 1175
66.7 TMouseEvent . . . 1176

67 Reference for unit 'Objects' 1177
67.1 Used units . . . 1177
67.2 Overview . . . 1177
67.3 Constants, types and variables . . . 1177
67.3.1 Constants . . . 1177
67.3.2 Types . . . 1180
67.3.3 Variables . . . 1182
67.4 Procedures and functions . . . 1182
67.4.1 Abstract . . . 1182
67.4.2 CallPointerConstructor . . . 1183
67.4.3 CallPointerLocal . . . 1183
67.4.4 CallPointerMethod . . . 1183
67.4.5 CallPointerMethodLocal . . . 1184
67.4.6 CallVoidConstructor . . . 1184
67.4.7 CallVoidLocal . . . 1184
67.4.8 CallVoidMethod . . . 1185
67.4.9 CallVoidMethodLocal . . . 1185
67.4.10 DisposeStr . . . 1185
67.4.11 LongDiv . . . 1185
67.4.12 LongMul . . . 1186
67.4.13 NewStr . . . 1186
67.4.14 RegisterObjects . . . 1187
67.4.15 RegisterType . . . 1187
67.4.16 SetStr . . . 1189
67.5 LongRec . . . 1189
67.6 PtrRec . . . 1189
67.7 TStreamRec . . . 1189
67.8 TStrIndexRec . . . 1190
67.9 WordRec . . . 1190

67.10 TBufStream 1190
67.10.1 Description 1190
67.10.2 Method overview 1191
67.10.3 TBufStream.Init 1191
67.10.4 TBufStream.Done 1191
67.10.5 TBufStream.Close 1192
67.10.6 TBufStream.Flush 1192
67.10.7 TBufStream.Truncate 1193
67.10.8 TBufStream.Seek 1193
67.10.9 TBufStream.Open 1193
67.10.10 TBufStream.Read 1194
67.10.11 TBufStream.Write 1194
67.11 TCollection 1194
67.11.1 Description 1194
67.11.2 Method overview 1195
67.11.3 TCollection.Init 1195
67.11.4 TCollection.Load 1195
67.11.5 TCollection.Done 1196
67.11.6 TCollection.At 1196
67.11.7 TCollection.IndexOf 1197
67.11.8 TCollection.GetItem 1198
67.11.9 TCollection.LastThat 1198
67.11.10 TCollection.FirstThat 1199
67.11.11 TCollection.Pack 1200
67.11.12 TCollection.FreeAll 1201
67.11.13 TCollection.DeleteAll 1202
67.11.14 TCollection.Free 1202
67.11.15 TCollection.Insert 1203
67.11.16 TCollection.Delete 1203
67.11.17 TCollection.AtFree 1204
67.11.18 TCollection.FreeItem 1205
67.11.19 TCollection.AtDelete 1205
67.11.20 TCollection.ForEach 1206
67.11.21 TCollection.SetLimit 1207
67.11.22 TCollection.Error 1207
67.11.23 TCollection.AtPut 1207
67.11.24 TCollection.AtInsert 1208
67.11.25 TCollection.Store 1209
67.11.26 TCollection.PutItem 1209
67.12 TDosStream 1209

67.12.1 Description . . . 1209
67.12.2 Method overview . . . 1209
67.12.3 TDosStream.Init . . . 1210
67.12.4 TDosStream.Done . . . 1210
67.12.5 TDosStream.Close . . . 1210
67.12.6 TDosStream.Truncate . . . 1211
67.12.7 TDosStream.Seek . . . 1211
67.12.8 TDosStream.Open . . . 1212
67.12.9 TDosStream.Read . . . 1213
67.12.10 TDosStream.Write . . . 1213
67.13 TMemoryStream . . . 1214
67.13.1 Description . . . 1214
67.13.2 Method overview . . . 1214
67.13.3 TMemoryStream.Init . . . 1214
67.13.4 TMemoryStream.Done . . . 1214
67.13.5 TMemoryStream.Truncate . . . 1215
67.13.6 TMemoryStream.Read . . . 1215
67.13.7 TMemoryStream.Write . . . 1216
67.14 TObject . . . 1216
67.14.1 Description . . . 1216
67.14.2 Method overview . . . 1216
67.14.3 TObject.Init . . . 1216
67.14.4 TObject.Free . . . 1217
67.14.5 TObject.Is_Object . . . 1217
67.14.6 TObject.Done . . . 1217
67.15 TPoint . . . 1218
67.15.1 Description . . . 1218
67.16 TRect . . . 1218
67.16.1 Description . . . 1218
67.16.2 Method overview . . . 1218
67.16.3 TRect.Empty . . . 1218
67.16.4 TRect.Equals . . . 1219
67.16.5 TRect.Contains . . . 1220
67.16.6 TRect.Copy . . . 1220
67.16.7 TRect.Union . . . 1221
67.16.8 TRect.Intersect . . . 1221
67.16.9 TRect.Move . . . 1222
67.16.10 TRect.Grow . . . 1223
67.16.11 TRect.Assign . . . 1223
67.17 TResourceCollection . . . 1224

67.17.1 Description . . . 1224
67.17.2 Method overview . . . 1224
67.17.3 TResourceCollection.KeyOf . . . 1224
67.17.4 TResourceCollection.GetItem . . . 1225
67.17.5 TResourceCollection.FreeItem . . . 1225
67.17.6 TResourceCollection.PutItem . . . 1225
67.18 TResourceFile . . . 1225
67.18.1 Description . . . 1225
67.18.2 Method overview . . . 1226
67.18.3 TResourceFile.Init . . . 1226
67.18.4 TResourceFile.Done . . . 1226
67.18.5 TResourceFile.Count . . . 1226
67.18.6 TResourceFile.KeyAt . . . 1227
67.18.7 TResourceFile.Get . . . 1227
67.18.8 TResourceFile.SwitchTo . . . 1227
67.18.9 TResourceFile.Flush . . . 1227
67.18.10 TResourceFile.Delete . . . 1228
67.18.11 TResourceFile.Put . . . 1228
67.19 TSortedCollection . . . 1228
67.19.1 Description . . . 1228
67.19.2 Method overview . . . 1229
67.19.3 TSortedCollection.Init . . . 1229
67.19.4 TSortedCollection.Load . . . 1229
67.19.5 TSortedCollection.KeyOf . . . 1229
67.19.6 TSortedCollection.IndexOf . . . 1230
67.19.7 TSortedCollection.Compare . . . 1230
67.19.8 TSortedCollection.Search . . . 1231
67.19.9 TSortedCollection.Insert . . . 1232
67.19.10 TSortedCollection.Store . . . 1233
67.20 TStrCollection . . . 1234
67.20.1 Description . . . 1234
67.20.2 Method overview . . . 1234
67.20.3 TStrCollection.Compare . . . 1234
67.20.4 TStrCollection.GetItem . . . 1235
67.20.5 TStrCollection.FreeItem . . . 1235
67.20.6 TStrCollection.PutItem . . . 1235
67.21 TStream . . . 1236
67.21.1 Description . . . 1236
67.21.2 Method overview . . . 1236
67.21.3 TStream.Init . . . 1236

67.21.4 TStream.Get . . . 1236
67.21.5 TStream.StrRead . . . 1237
67.21.6 TStream.GetPos . . . 1238
67.21.7 TStream.GetSize . . . 1238
67.21.8 TStream.ReadStr . . . 1239
67.21.9 TStream.Open . . . 1240
67.21.10 TStream.Close . . . 1240
67.21.11 TStream.Reset . . . 1240
67.21.12 TStream.Flush . . . 1241
67.21.13 TStream.Truncate . . . 1241
67.21.14 TStream.Put . . . 1241
67.21.15 TStream.StrWrite . . . 1242
67.21.16 TStream.WriteStr . . . 1242
67.21.17 TStream.Seek . . . 1242
67.21.18 TStream.Error . . . 1242
67.21.19 TStream.Read . . . 1243
67.21.20 TStream.Write . . . 1243
67.21.21 TStream.CopyFrom . . . 1244
67.22 TStringCollection . . . 1244
67.22.1 Description . . . 1244
67.22.2 Method overview . . . 1245
67.22.3 TStringCollection.GetItem . . . 1245
67.22.4 TStringCollection.Compare . . . 1245
67.22.5 TStringCollection.FreeItem . . . 1246
67.22.6 TStringCollection.PutItem . . . 1246
67.23 TStringList . . . 1246
67.23.1 Description . . . 1246
67.23.2 Method overview . . . 1247
67.23.3 TStringList.Load . . . 1247
67.23.4 TStringList.Done . . . 1247
67.23.5 TStringList.Get . . . 1247
67.24 TStrListMaker . . . 1248
67.24.1 Description . . . 1248
67.24.2 Method overview . . . 1248
67.24.3 TStrListMaker.Init . . . 1248
67.24.4 TStrListMaker.Done . . . 1248
67.24.5 TStrListMaker.Put . . . 1248
67.24.6 TStrListMaker.Store . . . 1249
67.25 TUnSortedStrCollection . . . 1249
67.25.1 Description . . . 1249

67.25.2 Method overview . . . 1249
67.25.3 TUnSortedStrCollection.Insert . . . 1249

68 Reference for unit 'objpas' 1251
68.1 Used units . . . 1251
68.2 Overview . . . 1251
68.3 Constants, types and variables . . . 1251
68.3.1 Constants . . . 1251
68.3.2 Types . . . 1251

69 Reference for unit 'ports' 1253
69.1 Used units . . . 1253
69.2 Overview . . . 1253
69.3 Constants, types and variables . . . 1253
69.3.1 Variables . . . 1253
69.4 tport . . . 1254
69.4.1 Description . . . 1254
69.4.2 Property overview . . . 1254
69.4.3 tport.pp . . . 1254
69.5 tportl . . . 1255
69.5.1 Description . . . 1255
69.5.2 Property overview . . . 1255
69.5.3 tportl.pp . . . 1255
69.6 tportw . . . 1255
69.6.1 Description . . . 1255
69.6.2 Property overview . . . 1255
69.6.3 tportw.pp . . . 1255

70 Reference for unit 'printer' 1256
70.1 Used units . . . 1256
70.2 Overview . . . 1256
70.3 Constants, types and variables . . . 1256
70.3.1 Variables . . . 1256
70.4 Procedures and functions . . . 1256
70.4.1 AssignLst . . . 1256
70.4.2 InitPrinter . . . 1257
70.4.3 IsLstAvailable . . . 1257

71 Reference for unit 'sharemem' 1258
71.1 Used units . . . 1258
71.2 Overview . . . 1258

72 Reference for unit 'Sockets' 1259
72.1 Used units . . . 1259
72.2 Overview . . . 1259
72.3 Constants, types and variables . . . 1259
72.3.1 Constants . . . 1259
72.3.2 Types . . . 1286
72.4 Procedures and functions . . . 1290
72.4.1 Accept . . . 1290
72.4.2 Bind . . . 1291
72.4.3 CloseSocket . . . 1291
72.4.4 Connect . . . 1292
72.4.5 fpaccept . . . 1294
72.4.6 fpbind . . . 1295
72.4.7 fpconnect . . . 1295
72.4.8 fpgetpeername . . . 1297
72.4.9 fpgetsockname . . . 1298
72.4.10 fpgetsockopt . . . 1298
72.4.11 fplisten . . . 1299
72.4.12 fprecv . . . 1299
72.4.13 fprecvfrom . . . 1300
72.4.14 fpsend . . . 1300
72.4.15 fpsendto . . . 1300
72.4.16 fpsetsockopt . . . 1301
72.4.17 fpshutdown . . . 1301
72.4.18 fpsocket . . . 1302
72.4.19 fpsocketpair . . . 1302
72.4.20 HostAddrToStr . . . 1302
72.4.21 HostAddrToStr6 . . . 1303
72.4.22 HostToNet . . . 1303
72.4.23 htonl . . . 1303
72.4.24 htons . . . 1303
72.4.25 NetAddrToStr . . . 1304
72.4.26 NetAddrToStr6 . . . 1304
72.4.27 NetToHost . . . 1304
72.4.28 NToHl . . . 1304
72.4.29 NToHs . . . 1305
72.4.30 ShortHostToNet . . . 1305
72.4.31 ShortNetToHost . . . 1305
72.4.32 Sock2File . . . 1305
72.4.33 Sock2Text . . . 1306

72.4.34 socketerror . . . 1306
72.4.35 Str2UnixSockAddr . . . 1306
72.4.36 StrToHostAddr . . . 1306
72.4.37 StrToHostAddr6 . . . 1307
72.4.38 StrToNetAddr . . . 1307
72.4.39 StrToNetAddr6 . . . 1307
72.4.40 TryStrToHostAddr . . . 1307
72.4.41 TryStrToHostAddr6 . . . 1308
72.5 linger . . . 1308
72.6 sockaddr_in . . . 1308
72.7 sockaddr_in6 . . . 1308
72.8 sockaddr_un . . . 1309
72.9 TUnixSockAddr . . . 1309
72.10 ucred . . . 1309

73 Reference for unit 'Strings' 1310
73.1 Used units . . . 1310
73.2 Overview . . . 1310
73.3 Procedures and functions . . . 1310
73.3.1 stralloc . . . 1310
73.3.2 strcat . . . 1310
73.3.3 strcomp . . . 1311
73.3.4 strcopy . . . 1311
73.3.5 strdispose . . . 1312
73.3.6 strecopy . . . 1313
73.3.7 strend . . . 1313
73.3.8 stricomp . . . 1314
73.3.9 stripos . . . 1314
73.3.10 striscan . . . 1315
73.3.11 strlcat . . . 1315
73.3.12 strlcomp . . . 1316
73.3.13 strlcopy . . . 1316
73.3.14 strlen . . . 1317
73.3.15 strlicomp . . . 1317
73.3.16 strlower . . . 1318
73.3.17 strmove . . . 1318
73.3.18 strnew . . . 1319
73.3.19 strpas . . . 1320
73.3.20 strpcopy . . . 1320
73.3.21 strpos . . . 1321

73.3.22 strriscan . . . 1321
73.3.23 strrscan . . . 1322
73.3.24 strscan . . . 1322
73.3.25 strupper . . . 1322

74 Reference for unit 'StrUtils' 1323
74.1 Used units . . . 1323
74.2 Constants, types and variables . . . 1323
74.2.1 Resource strings . . . 1323
74.2.2 Constants . . . 1323
74.2.3 Types . . . 1324
74.3 Procedures and functions . . . 1326
74.3.1 AddChar . . . 1326
74.3.2 AddCharR . . . 1326
74.3.3 AnsiContainsStr . . . 1326
74.3.4 AnsiContainsText . . . 1327
74.3.5 AnsiEndsStr . . . 1327
74.3.6 AnsiEndsText . . . 1327
74.3.7 AnsiIndexStr . . . 1327
74.3.8 AnsiIndexText . . . 1328
74.3.9 AnsiLeftStr . . . 1328
74.3.10 AnsiMatchStr . . . 1328
74.3.11 AnsiMatchText . . . 1329
74.3.12 AnsiMidStr . . . 1329
74.3.13 AnsiProperCase . . . 1329
74.3.14 AnsiReplaceStr . . . 1329
74.3.15 AnsiReplaceText . . . 1330
74.3.16 AnsiResemblesText . . . 1330
74.3.17 AnsiReverseString . . . 1330
74.3.18 AnsiRightStr . . . 1331
74.3.19 AnsiStartsStr . . . 1331
74.3.20 AnsiStartsText . . . 1331
74.3.21 BinToHex . . . 1331
74.3.22 ContainsStr . . . 1332
74.3.23 ContainsText . . . 1332
74.3.24 Copy2Space . . . 1333
74.3.25 Copy2SpaceDel . . . 1333
74.3.26 Copy2Symb . . . 1333
74.3.27 Copy2SymbDel . . . 1334
74.3.28 Dec2Numb . . . 1334

74.3.29 DecodeSoundexInt . . . 1334
74.3.30 DecodeSoundexWord . . . 1334
74.3.31 DelChars . . . 1335
74.3.32 DelSpace . . . 1335
74.3.33 DelSpace1 . . . 1335
74.3.34 DupeString . . . 1335
74.3.35 EndsStr . . . 1336
74.3.36 EndsText . . . 1336
74.3.37 ExtractDelimited . . . 1336
74.3.38 ExtractSubstr . . . 1337
74.3.39 ExtractWord . . . 1337
74.3.40 ExtractWordPos . . . 1337
74.3.41 FindMatchesBoyerMooreCaseInSensitive . . . 1338
74.3.42 FindMatchesBoyerMooreCaseSensitive . . . 1338
74.3.43 FindPart . . . 1339
74.3.44 GetCmdLineArg . . . 1339
74.3.45 Hex2Dec . . . 1340
74.3.46 Hex2Dec64 . . . 1340
74.3.47 HexToBin . . . 1340
74.3.48 IfThen . . . 1340
74.3.49 in(string,):Boolean . . . 1341
74.3.50 in(UnicodeString,):Boolean . . . 1341
74.3.51 IndexStr . . . 1341
74.3.52 IndexText . . . 1342
74.3.53 IntToBin . . . 1342
74.3.54 IntToRoman . . . 1342
74.3.55 IsEmptyStr . . . 1342
74.3.56 IsWild . . . 1343
74.3.57 IsWordPresent . . . 1343
74.3.58 LeftBStr . . . 1343
74.3.59 LeftStr . . . 1344
74.3.60 MatchStr . . . 1344
74.3.61 MatchText . . . 1344
74.3.62 MidBStr . . . 1345
74.3.63 MidStr . . . 1345
74.3.64 NaturalCompareText . . . 1345
74.3.65 NPos . . . 1346
74.3.66 Numb2Dec . . . 1346
74.3.67 Numb2USA . . . 1346
74.3.68 PadCenter . . . 1347

74.3.69 PadLeft . . . 1347
74.3.70 PadRight . . . 1347
74.3.71 PosEx . . . 1347
74.3.72 PosSet . . . 1348
74.3.73 PosSetEx . . . 1348
74.3.74 RandomFrom . . . 1348
74.3.75 RemoveLeadingchars . . . 1349
74.3.76 RemovePadChars . . . 1349
74.3.77 RemoveTrailingChars . . . 1349
74.3.78 ReplaceStr . . . 1350
74.3.79 ReplaceText . . . 1350
74.3.80 ResemblesText . . . 1350
74.3.81 ReverseString . . . 1350
74.3.82 RightBStr . . . 1351
74.3.83 RightStr . . . 1351
74.3.84 RomanToInt . . . 1351
74.3.85 RomanToIntDef . . . 1352
74.3.86 RPos . . . 1352
74.3.87 RPosEx . . . 1352
74.3.88 SearchBuf . . . 1353
74.3.89 Soundex . . . 1353
74.3.90 SoundexCompare . . . 1354
74.3.91 SoundexInt . . . 1354
74.3.92 SoundexProc . . . 1354
74.3.93 SoundexSimilar . . . 1355
74.3.94 SoundexWord . . . 1355
74.3.95 SplitCommandLine . . . 1355
74.3.96 SplitString . . . 1356
74.3.97 StartsStr . . . 1356
74.3.98 StartsText . . . 1356
74.3.99 StringReplace . . . 1357
74.3.100 StringsReplace . . . 1357
74.3.101 StuffString . . . 1358
74.3.102 Tab2Space . . . 1358
74.3.103 TrimLeftSet . . . 1358
74.3.104 TrimRightSet . . . 1358
74.3.105 TrimSet . . . 1359
74.3.106 TryRomanToInt . . . 1359
74.3.107 WordCount . . . 1359
74.3.108 WordPosition . . . 1360

74.3.109 XorDecode . . . 1360
74.3.110 XorEncode . . . 1360
74.3.111 XorString . . . 1361

75 Reference for unit 'System' 1362
75.1 Overview . . . 1362
75.2 A list of managed types. . . . 1362
75.3 Unicode and codepage support. . . . 1363
75.4 Miscellaneous functions. . . . 1364
75.5 Operating System functions. . . . 1365
75.6 String handling. . . . 1365
75.7 Mathematical routines. . . . 1365
75.8 Memory management functions. . . . 1366
75.9 File handling functions. . . . 1367
75.10 Run-Time Error behaviour. . . . 1368
75.11 Constants, types and variables . . . 1368
75.11.1 Constants . . . 1368
75.11.2 Types . . . 1396
75.11.3 Variables . . . 1441
75.12 Procedures and functions . . . 1446
75.12.1 Abs . . . 1446
75.12.2 AbstractError . . . 1447
75.12.3 AcquireExceptionObject . . . 1447
75.12.4 add(variant,variant):variant . . . 1447
75.12.5 AddExitProc . . . 1447
75.12.6 Addr . . . 1448
75.12.7 Align . . . 1448
75.12.8 AllocMem . . . 1449
75.12.9 AnsiToUtf8 . . . 1449
75.12.10 Append . . . 1449
75.12.11 ArcTan . . . 1450
75.12.12 ArrayStringToPPchar . . . 1450
75.12.13 Assert . . . 1451
75.12.14 Assign . . . 1451
75.12.15 assign(Comp):olevariant . . . 1452
75.12.16 assign(Comp):variant . . . 1452
75.12.17 assign(extended):olevariant . . . 1452
75.12.18 assign(extended):variant . . . 1452
75.12.19 assign(NativeInt):variant . . . 1452
75.12.20 assign(NativeUInt):variant . . . 1453

75.12.21 assign(olevariant):Comp . . . 1453
75.12.22 assign(olevariant):extended . . . 1453
75.12.23 assign(olevariant):Real . . . 1453
75.12.24 assign(olevariant):single . . . 1453
75.12.25 assign(olevariant):UnicodeString . . . 1453
75.12.26 assign(Real):olevariant . . . 1453
75.12.27 assign(Real):variant . . . 1454
75.12.28 assign(Real48):extended . . . 1454
75.12.29 assign(single):olevariant . . . 1454
75.12.30 assign(single):variant . . . 1454
75.12.31 assign(UCS4String):variant . . . 1454
75.12.32 assign(UnicodeString):olevariant . . . 1454
75.12.33 assign(UnicodeString):variant . . . 1455
75.12.34 assign(UTF8String):variant . . . 1455
75.12.35 assign(variant):Comp . . . 1455
75.12.36 assign(variant):extended . . . 1455
75.12.37 assign(variant):NativeInt . . . 1455
75.12.38 assign(variant):NativeUInt . . . 1455
75.12.39 assign(variant):Real . . . 1455
75.12.40 assign(variant):single . . . 1456
75.12.41 assign(variant):unicodestring . . . 1456
75.12.42 assign(variant):UTF8String . . . 1456
75.12.43 Assigned . . . 1456
75.12.44 BasicEventCreate . . . 1457
75.12.45 BasicEventDestroy . . . 1457
75.12.46 BasicEventResetEvent . . . 1457
75.12.47 BasicEventSetEvent . . . 1457
75.12.48 BasicEventWaitFor . . . 1457
75.12.49 BeginThread . . . 1458
75.12.50 BEtoN . . . 1458
75.12.51 BinStr . . . 1458
75.12.52 BlockRead . . . 1459
75.12.53 BlockWrite . . . 1460
75.12.54 Break . . . 1460
75.12.55 BsfByte . . . 1462
75.12.56 BsfDWord . . . 1462
75.12.57 BsfQWord . . . 1462
75.12.58 BsfWord . . . 1463
75.12.59 BsrByte . . . 1463
75.12.60 BsrDWord . . . 1463

75.12.61 BsrQWord . . . 1464
75.12.62 BsrWord . . . 1464
75.12.63 CaptureBacktrace . . . 1464
75.12.64 ChDir . . . 1464
75.12.65 Chr . . . 1465
75.12.66 Close . . . 1465
75.12.67 CloseThread . . . 1466
75.12.68 CompareByte . . . 1466
75.12.69 CompareChar . . . 1467
75.12.70 CompareChar0 . . . 1469
75.12.71 CompareDWord . . . 1469
75.12.72 CompareWord . . . 1470
75.12.73 Concat . . . 1471
75.12.74 Continue . . . 1472
75.12.75 Copy . . . 1473
75.12.76 CopyArray . . . 1474
75.12.77 Cos . . . 1474
75.12.78 CSeg . . . 1475
75.12.79 Dec . . . 1475
75.12.80 Default . . . 1476
75.12.81 DefaultAnsi2UnicodeMove . . . 1476
75.12.82 DefaultAnsi2WideMove . . . 1477
75.12.83 DefaultUnicode2AnsiMove . . . 1477
75.12.84 Delete . . . 1477
75.12.85 Dispose . . . 1478
75.12.86 divide(variant,variant):variant . . . 1479
75.12.87 DoneCriticalSection . . . 1479
75.12.88 DoneThread . . . 1480
75.12.89 DSeg . . . 1480
75.12.90 DumpExceptionBacktrace . . . 1480
75.12.91 Dump_Stack . . . 1481
75.12.92 DynArrayBounds . . . 1481
75.12.93 DynArrayClear . . . 1481
75.12.94 DynArrayDim . . . 1481
75.12.95 DynArrayIndex . . . 1482
75.12.96 DynArraySetLength . . . 1482
75.12.97 DynArraySize . . . 1482
75.12.98 EmptyMethod . . . 1482
75.12.99 EndThread . . . 1483
75.12.100EnterCriticalSection . . . 1483

75.12.101EnumResourceLanguages 1483
75.12.102EnumResourceNames 1484
75.12.103EnumResourceTypes 1484
75.12.104EOF 1484
75.12.105EOLn 1485
75.12.106equal(variant,variant):Boolean 1486
75.12.107Erase 1486
75.12.108Error 1487
75.12.109Exclude 1487
75.12.110Exit 1488
75.12.111Exp 1490
75.12.112Fail 1490
75.12.113FilePos 1491
75.12.114FileSize 1492
75.12.115FillByte 1493
75.12.116FillChar 1493
75.12.117FillDWord 1494
75.12.118FillWord 1494
75.12.119Finalize 1495
75.12.120FinalizeArray 1495
75.12.121FindResource 1496
75.12.122FindResourceEx 1496
75.12.123float_raise 1497
75.12.124Flush 1497
75.12.125FlushThread 1498
75.12.126FMADouble 1498
75.12.127FMAExtended 1498
75.12.128FMASingle 1498
75.12.129FPower10 1498
75.12.130Frac 1499
75.12.131FreeLibrary 1499
75.12.132Freemem 1499
75.12.133Freememory 1500
75.12.134FreeResource 1500
75.12.135Get8087CW 1500
75.12.136GetCPUCount 1500
75.12.137GetCurrentThreadId 1501
75.12.138GetDir 1501
75.12.139GetDynLibsManager 1502
75.12.140GetFPCHeapStatus 1502

75.12.141GetHeapStatus . . . 1502
75.12.142GetLoadErrorStr . . . 1502
75.12.143GetMem . . . 1502
75.12.144GetMemory . . . 1503
75.12.145GetMemoryManager . . . 1503
75.12.146GetMXCSR . . . 1503
75.12.147GetProcAddress . . . 1503
75.12.148GetProcedureAddress . . . 1504
75.12.149GetProcessID . . . 1504
75.12.150GetResourceManager . . . 1504
75.12.151GetSSECSR . . . 1504
75.12.152GetTextCodePage . . . 1505
75.12.153GetThreadID . . . 1505
75.12.154GetThreadManager . . . 1505
75.12.155GetTypeKind . . . 1505
75.12.156GetUnicodeStringManager . . . 1506
75.12.157GetVariantManager . . . 1506
75.12.158GetWideStringManager . . . 1506
75.12.159get_caller_addr . . . 1507
75.12.160get_caller_frame . . . 1507
75.12.161get_caller_stackinfo . . . 1507
75.12.162get_cmdline . . . 1507
75.12.163get_frame . . . 1508
75.12.164Get_pc_addr . . . 1508
75.12.165greaterthan(variant,variant):Boolean . . . 1508
75.12.166greaterthanorequal(variant,variant):Boolean . . . 1508
75.12.167Halt . . . 1509
75.12.168HexStr . . . 1509
75.12.169Hi . . . 1510
75.12.170High . . . 1511
75.12.171HINSTANCE . . . 1512
75.12.172Inc . . . 1512
75.12.173Include . . . 1513
75.12.174IndexByte . . . 1513
75.12.175IndexChar . . . 1514
75.12.176IndexChar0 . . . 1515
75.12.177IndexDWord . . . 1515
75.12.178IndexQWord . . . 1516
75.12.179Indexword . . . 1516
75.12.180InitCriticalSection . . . 1517

75.12.181Initialize . . . 1517
75.12.182InitializeArray . . . 1520
75.12.183InitThread . . . 1520
75.12.184InitThreadVars . . . 1520
75.12.185Insert . . . 1521
75.12.186Int . . . 1521
75.12.187intdivide(variant,variant):variant . . . 1522
75.12.188InterlockedCompareExchange . . . 1522
75.12.189InterlockedCompareExchange64 . . . 1523
75.12.190InterlockedCompareExchangePointer . . . 1523
75.12.191InterlockedDecrement . . . 1523
75.12.192InterlockedDecrement64 . . . 1523
75.12.193InterlockedExchange . . . 1523
75.12.194InterlockedExchange64 . . . 1524
75.12.195InterlockedExchangeAdd . . . 1524
75.12.196InterlockedExchangeAdd64 . . . 1524
75.12.197InterlockedIncrement . . . 1525
75.12.198InterlockedIncrement64 . . . 1525
75.12.199IOResult . . . 1525
75.12.200IsDynArrayRectangular . . . 1526
75.12.201IsMemoryManagerSet . . . 1527
75.12.202Is_IntResource . . . 1527
75.12.203KillThread . . . 1527
75.12.204LazyInitThreading . . . 1527
75.12.205LeaveCriticalSection . . . 1527
75.12.206leftshift(variant,variant):variant . . . 1528
75.12.207Length . . . 1528
75.12.208lessthan(variant,variant):Boolean . . . 1529
75.12.209lessthanorequal(variant,variant):Boolean . . . 1529
75.12.210LEtoN . . . 1529
75.12.211Ln . . . 1530
75.12.212Lo . . . 1530
75.12.213LoadLibrary . . . 1531
75.12.214LoadResource . . . 1531
75.12.215LockResource . . . 1532
75.12.216logicaland(variant,variant):variant . . . 1532
75.12.217logicalnot(variant):variant . . . 1532
75.12.218logicalor(variant,variant):variant . . . 1533
75.12.219logicalxor(variant,variant):variant . . . 1533
75.12.220longjmp . . . 1533

75.12.221Low 1534
75.12.222LowerCase 1534
75.12.223MakeLangID 1535
75.12.224MemSize 1535
75.12.225MkDir 1535
75.12.226modulus(variant,variant):variant 1535
75.12.227Move 1536
75.12.228MoveChar0 1536
75.12.229multiply(variant,variant):variant 1537
75.12.230negative(variant):variant 1537
75.12.231New 1537
75.12.232NtoBE 1538
75.12.233NtoLE 1538
75.12.234Null 1538
75.12.235OctStr 1539
75.12.236Odd 1539
75.12.237Ofs 1540
75.12.238Ord 1540
75.12.239Pack 1541
75.12.240ParamCount 1541
75.12.241ParamStr 1542
75.12.242Pi 1542
75.12.243PopCnt 1543
75.12.244Pos 1543
75.12.245power(variant,variant):variant 1544
75.12.246Pred 1545
75.12.247Prefetch 1545
75.12.248Ptr 1545
75.12.249RaiseList 1546
75.12.250Random 1546
75.12.251Randomize 1547
75.12.252Read 1547
75.12.253ReadBarrier 1548
75.12.254ReadDependencyBarrier 1548
75.12.255ReadLn 1549
75.12.256ReadStr 1549
75.12.257ReadWriteBarrier 1550
75.12.258Real2Double 1550
75.12.259ReAllocMem 1551
75.12.260ReAllocMemory 1551

75.12.261RegisterLazyInitThreadingProc 1551
75.12.262ReleaseExceptionObject 1551
75.12.263Rename 1552
75.12.264Reset 1552
75.12.265ResumeThread 1553
75.12.266Rewrite 1553
75.12.267rightshift(variant,variant):variant 1554
75.12.268RmDir 1554
75.12.269RolByte 1555
75.12.270RolDWord 1555
75.12.271RolQWord 1556
75.12.272RolWord 1556
75.12.273RorByte 1556
75.12.274RorDWord 1557
75.12.275RorQWord 1557
75.12.276RorWord 1557
75.12.277Round 1557
75.12.278RTLEventCreate 1559
75.12.279RTLEventDestroy 1559
75.12.280RTLEventResetEvent 1559
75.12.281RTLEventSetEvent 1560
75.12.282RTLEventWaitFor 1560
75.12.283RunError 1560
75.12.284SafeLoadLibrary 1561
75.12.285SarInt64 1561
75.12.286SarLongint 1561
75.12.287SarShortint 1562
75.12.288SarSmallint 1562
75.12.289Seek 1562
75.12.290SeekEOF 1563
75.12.291SeekEOLn 1564
75.12.292Seg 1564
75.12.293Set8087CW 1565
75.12.294SetCodePage 1565
75.12.295SetDynLibsManager 1565
75.12.296Setjmp 1565
75.12.297SetLength 1566
75.12.298SetMemoryManager 1567
75.12.299SetMultiByteConversionCodePage 1568
75.12.300SetMultiByteFileSystemCodePage 1568

75.12.301SetMultiByteRTLFileSystemCodePage 1568
75.12.302SetMXCSR 1569
75.12.303SetResourceManager 1569
75.12.304SetSSECSR 1569
75.12.305SetString 1569
75.12.306SetTextBuf 1570
75.12.307SetTextCodePage 1571
75.12.308SetTextLineEnding 1571
75.12.309SetThreadDebugName 1571
75.12.310SetThreadManager 1572
75.12.311SetUnicodeStringManager 1572
75.12.312SetVariantManager 1572
75.12.313SetWideStringManager 1573
75.12.314ShortCompareText 1573
75.12.315Sin 1573
75.12.316SizeOf 1574
75.12.317SizeofResource 1574
75.12.318Slice 1575
75.12.319Space 1575
75.12.320SPtr 1575
75.12.321Sqr 1576
75.12.322Sqrt 1576
75.12.323SSeg 1577
75.12.324StackTop 1577
75.12.325Str 1577
75.12.326StringCodePage 1579
75.12.327StringElementSize 1579
75.12.328StringOfChar 1580
75.12.329StringRefCount 1580
75.12.330StringToPPChar 1580
75.12.331StringToUnicodeChar 1581
75.12.332StringToWideChar 1581
75.12.333StrLen 1582
75.12.334StrPas 1582
75.12.335subtract(variant,variant):variant 1582
75.12.336Succ 1582
75.12.337SuspendThread 1583
75.12.338Swap 1583
75.12.339SwapEndian 1584
75.12.340SysAllocMem 1584

75.12.341SysAssert . . . 1584
75.12.342SysBacktraceStr . . . 1584
75.12.343SysFlushStdIO . . . 1585
75.12.344SysFreemem . . . 1585
75.12.345SysFreememSize . . . 1585
75.12.346SysGetFPCHeapStatus . . . 1585
75.12.347SysGetHeapStatus . . . 1586
75.12.348SysGetmem . . . 1586
75.12.349SysInitExceptions . . . 1586
75.12.350SysInitFPU . . . 1586
75.12.351SysInitStdIO . . . 1586
75.12.352SysMemSize . . . 1587
75.12.353SysReAllocMem . . . 1587
75.12.354SysResetFPU . . . 1587
75.12.355SysSetCtrlBreakHandler . . . 1587
75.12.356SysTryResizeMem . . . 1587
75.12.357ThreadGetPriority . . . 1588
75.12.358ThreadSetPriority . . . 1588
75.12.359ThreadSwitch . . . 1588
75.12.360ToSingleByteFileSystemEncodedFileName . . . 1588
75.12.361Trunc . . . 1589
75.12.362Truncate . . . 1590
75.12.363TryEnterCriticalSection . . . 1591
75.12.364TypeInfo . . . 1591
75.12.365TypeOf . . . 1591
75.12.366UCS4StringToUnicodeString . . . 1592
75.12.367UCS4StringToWideString . . . 1592
75.12.368Unassigned . . . 1592
75.12.369UnicodeCharLenToString . . . 1592
75.12.370UnicodeCharLenToStrVar . . . 1592
75.12.371UnicodeCharToString . . . 1593
75.12.372UnicodeCharToStrVar . . . 1593
75.12.373UnicodeStringToUCS4String . . . 1593
75.12.374UnicodeToUtf8 . . . 1594
75.12.375UniqueString . . . 1594
75.12.376UnloadLibrary . . . 1594
75.12.377UnlockResource . . . 1594
75.12.378UnPack . . . 1595
75.12.379UpCase . . . 1595
75.12.380Utf8CodePointLen . . . 1596

75.12.381UTF8Decode . . . 1596
75.12.382UTF8Encode . . . 1596
75.12.383Utf8ToAnsi . . . 1597
75.12.384UTF8ToString . . . 1597
75.12.385Utf8ToUnicode . . . 1597
75.12.386Val . . . 1598
75.12.387VarArrayGet . . . 1598
75.12.388VarArrayPut . . . 1599
75.12.389VarArrayRedim . . . 1599
75.12.390VarCast . . . 1600
75.12.391WaitForThreadTerminate . . . 1600
75.12.392WideCharLenToString . . . 1600
75.12.393WideCharLenToStrVar . . . 1600
75.12.394WideCharToString . . . 1601
75.12.395WideCharToStrVar . . . 1601
75.12.396WideStringToUCS4String . . . 1601
75.12.397Write . . . 1602
75.12.398WriteBarrier . . . 1603
75.12.399WriteLn . . . 1603
75.12.400WriteStr . . . 1604
75.13 TDoubleRec . . . 1604
75.13.1 Method overview . . . 1605
75.13.2 Property overview . . . 1605
75.13.3 TDoubleRec.Mantissa . . . 1605
75.13.4 TDoubleRec.Fraction . . . 1606
75.13.5 TDoubleRec.Exponent . . . 1606
75.13.6 TDoubleRec.SpecialType . . . 1606
75.13.7 TDoubleRec.BuildUp . . . 1606
75.13.8 TDoubleRec.Sign . . . 1607
75.13.9 TDoubleRec.Exp . . . 1607
75.13.10 TDoubleRec.Frac . . . 1607
75.14 TExtended80Rec . . . 1607
75.14.1 Method overview . . . 1608
75.14.2 Property overview . . . 1608
75.14.3 TExtended80Rec.Mantissa . . . 1608
75.14.4 TExtended80Rec.Fraction . . . 1609
75.14.5 TExtended80Rec.Exponent . . . 1609
75.14.6 TExtended80Rec.SpecialType . . . 1609
75.14.7 TExtended80Rec.BuildUp . . . 1609
75.14.8 TExtended80Rec.Sign . . . 1610

75.14.9 TExtended80Rec.Exp . . . 1610
75.15 tinterfaceentry . . . 1610
75.15.1 Property overview . . . 1611
75.15.2 tinterfaceentry.IID . . . 1611
75.15.3 tinterfaceentry.IIDStr . . . 1611
75.16 TSingleRec . . . 1611
75.16.1 Method overview . . . 1612
75.16.2 Property overview . . . 1612
75.16.3 TSingleRec.Mantissa . . . 1612
75.16.4 TSingleRec.Fraction . . . 1613
75.16.5 TSingleRec.Exponent . . . 1613
75.16.6 TSingleRec.SpecialType . . . 1613
75.16.7 TSingleRec.BuildUp . . . 1613
75.16.8 TSingleRec.Sign . . . 1613
75.16.9 TSingleRec.Exp . . . 1614
75.16.10 TSingleRec.Frac . . . 1614
75.17 TVmt . . . 1614
75.17.1 Property overview . . . 1615
75.17.2 TVmt.vParent . . . 1615
75.18 IDispatch . . . 1615
75.18.1 Description . . . 1615
75.18.2 Method overview . . . 1615
75.18.3 IDispatch.GetTypeInfoCount . . . 1616
75.18.4 IDispatch.GetTypeInfo . . . 1616
75.18.5 IDispatch.GetIDsOfNames . . . 1616
75.18.6 IDispatch.Invoke . . . 1616
75.19 IEnumerable . . . 1616
75.19.1 Description . . . 1616
75.19.2 Method overview . . . 1616
75.19.3 IEnumerable.GetEnumerator . . . 1617
75.20 IEnumerator . . . 1617
75.20.1 Description . . . 1617
75.20.2 Method overview . . . 1617
75.20.3 Property overview . . . 1618
75.20.4 IEnumerator.GetCurrent . . . 1618
75.20.5 IEnumerator.MoveNext . . . 1618
75.20.6 IEnumerator.Reset . . . 1618
75.20.7 IEnumerator.Current . . . 1618
75.21 IInvokable . . . 1619
75.21.1 Description . . . 1619

75.22 IUnknown . . . 1619
75.22.1 Description . . . 1619
75.22.2 Method overview . . . 1619
75.22.3 IUnknown.QueryInterface . . . 1619
75.22.4 IUnknown._AddRef . . . 1619
75.22.5 IUnknown._Release . . . 1620
75.23 TAggregatedObject . . . 1620
75.23.1 Description . . . 1620
75.23.2 Method overview . . . 1620
75.23.3 Property overview . . . 1620
75.23.4 TAggregatedObject.Create . . . 1620
75.23.5 TAggregatedObject.Controller . . . 1621
75.24 TContainedObject . . . 1621
75.24.1 Description . . . 1621
75.24.2 Interfaces overview . . . 1621
75.25 TInterfacedObject . . . 1621
75.25.1 Description . . . 1621
75.25.2 Interfaces overview . . . 1621
75.25.3 Method overview . . . 1621
75.25.4 Property overview . . . 1622
75.25.5 TInterfacedObject.destroy . . . 1622
75.25.6 TInterfacedObject.AfterConstruction . . . 1622
75.25.7 TInterfacedObject.BeforeDestruction . . . 1622
75.25.8 TInterfacedObject.NewInstance . . . 1622
75.25.9 TInterfacedObject.RefCount . . . 1623
75.26 TObject . . . 1623
75.26.1 Description . . . 1623
75.26.2 Method overview . . . 1624
75.26.3 TObject.Create . . . 1624
75.26.4 TObject.Destroy . . . 1625
75.26.5 TObject.newinstance . . . 1625
75.26.6 TObject.FreeInstance . . . 1625
75.26.7 TObject.SafeCallException . . . 1626
75.26.8 TObject.DefaultHandler . . . 1626
75.26.9 TObject.Free . . . 1626
75.26.10 TObject.InitInstance . . . 1626
75.26.11 TObject.CleanupInstance . . . 1627
75.26.12 TObject.ClassType . . . 1627
75.26.13 TObject.ClassInfo . . . 1627
75.26.14 TObject.ClassName . . . 1627

75.26.15 TObject.ClassNameIs . . . 1628
75.26.16 TObject.ClassParent . . . 1628
75.26.17 TObject.InstanceSize . . . 1628
75.26.18 TObject.InheritsFrom . . . 1628
75.26.19 TObject.StringMessageTable . . . 1629
75.26.20 TObject.MethodAddress . . . 1629
75.26.21 TObject.MethodName . . . 1629
75.26.22 TObject.FieldAddress . . . 1629
75.26.23 TObject.AfterConstruction . . . 1630
75.26.24 TObject.BeforeDestruction . . . 1630
75.26.25 TObject.DefaultHandlerStr . . . 1630
75.26.26 TObject.Dispatch . . . 1630
75.26.27 TObject.DispatchStr . . . 1631
75.26.28 TObject.GetInterface . . . 1631
75.26.29 TObject.GetInterfaceByStr . . . 1631
75.26.30 TObject.GetInterfaceWeak . . . 1632
75.26.31 TObject.GetInterfaceEntry . . . 1632
75.26.32 TObject.GetInterfaceEntryByStr . . . 1632
75.26.33 TObject.GetInterfaceTable . . . 1632
75.26.34 TObject.UnitName . . . 1633
75.26.35 TObject.QualifiedClassName . . . 1633
75.26.36 TObject.Equals . . . 1633
75.26.37 TObject.GetHashCode . . . 1633
75.26.38 TObject.ToString . . . 1634

76 Reference for unit 'sysutils' 1635
76.1 Used units . . . 1635
76.2 Overview . . . 1635
76.3 Type Helpers for basic types. . . . 1635
76.4 Localization support. . . . 1636
76.5 Unicode and codepage awareness. . . . 1636
76.6 Miscellaneous conversion routines. . . . 1637
76.7 Date/time routines. . . . 1638
76.8 File Name handling routines. . . . 1638
76.9 File input/output routines. . . . 1639
76.10 PChar related functions. . . . 1640
76.11 Date and time formatting characters. . . . 1641
76.12 Formatting strings. . . . 1642
76.13 String functions. . . . 1642
76.14 Constants, types and variables . . . 1643

76.14.1 Constants . . . 1643
76.14.2 Types . . . 1651
76.14.3 Variables . . . 1663
76.15 Procedures and functions . . . 1668
76.15.1 AbandonSignalHandler . . . 1668
76.15.2 Abort . . . 1668
76.15.3 AddDisk . . . 1668
76.15.4 AddTerminateProc . . . 1669
76.15.5 AdjustLineBreaks . . . 1669
76.15.6 AnsiCompareFileName . . . 1670
76.15.7 AnsiCompareStr . . . 1670
76.15.8 AnsiCompareText . . . 1671
76.15.9 AnsiDequotedStr . . . 1672
76.15.10 AnsiExtractQuotedStr . . . 1672
76.15.11 AnsiLastChar . . . 1673
76.15.12 AnsiLowerCase . . . 1674
76.15.13 AnsiLowerCaseFileName . . . 1674
76.15.14 AnsiPos . . . 1675
76.15.15 AnsiQuotedStr . . . 1675
76.15.16 AnsiSameStr . . . 1675
76.15.17 AnsiSameText . . . 1675
76.15.18 AnsiStrComp . . . 1676
76.15.19 AnsiStrIComp . . . 1676
76.15.20 AnsiStrLastChar . . . 1677
76.15.21 AnsiStrLComp . . . 1678
76.15.22 AnsiStrLIComp . . . 1679
76.15.23 AnsiStrLower . . . 1680
76.15.24 AnsiStrPos . . . 1680
76.15.25 AnsiStrRScan . . . 1681
76.15.26 AnsiStrScan . . . 1681
76.15.27 AnsiStrUpper . . . 1681
76.15.28 AnsiUpperCase . . . 1682
76.15.29 AnsiUpperCaseFileName . . . 1682
76.15.30 AppendStr . . . 1683
76.15.31 ApplicationName . . . 1683
76.15.32 ArrayOfConstToStr . . . 1684
76.15.33 ArrayOfConstToStrArray . . . 1684
76.15.34 AssignStr . . . 1684
76.15.35 BCDToInt . . . 1684
76.15.36 Beep . . . 1685

76.15.37 BoolToStr . . . 1685
76.15.38 ByteLength . . . 1686
76.15.39 BytesOf . . . 1686
76.15.40 ByteToCharIndex . . . 1686
76.15.41 ByteToCharLen . . . 1686
76.15.42 ByteType . . . 1687
76.15.43 CallTerminateProcs . . . 1687
76.15.44 ChangeFileExt . . . 1687
76.15.45 CharInSet . . . 1687
76.15.46 CharToByteLen . . . 1688
76.15.47 CheckOSError . . . 1688
76.15.48 CodePageNameToCodePage . . . 1688
76.15.49 CodePageToCodePageName . . . 1688
76.15.50 CompareMem . . . 1689
76.15.51 CompareMemRange . . . 1689
76.15.52 CompareStr . . . 1689
76.15.53 CompareText . . . 1690
76.15.54 ComposeDateTime . . . 1691
76.15.55 ConcatPaths . . . 1691
76.15.56 CreateDir . . . 1692
76.15.57 CreateGUID . . . 1693
76.15.58 CurrentYear . . . 1693
76.15.59 CurrToStr . . . 1693
76.15.60 CurrToStrF . . . 1694
76.15.61 Date . . . 1694
76.15.62 DateTimeToFileDate . . . 1695
76.15.63 DateTimeToStr . . . 1695
76.15.64 DateTimeToString . . . 1696
76.15.65 DateTimeToSystemTime . . . 1697
76.15.66 DateTimeToTimeStamp . . . 1697
76.15.67 DateToStr . . . 1698
76.15.68 DayOfWeek . . . 1699
76.15.69 DecodeDate . . . 1699
76.15.70 DecodeDateFully . . . 1700
76.15.71 DecodeTime . . . 1700
76.15.72 DeleteFile . . . 1700
76.15.73 DirectoryExists . . . 1701
76.15.74 DiskFree . . . 1701
76.15.75 DiskSize . . . 1702
76.15.76 DisposeStr . . . 1703

76.15.77 DoDirSeparators . . . 1703
76.15.78 EncodeDate . . . 1704
76.15.79 EncodeTime . . . 1704
76.15.80 ExceptAddr . . . 1705
76.15.81 ExceptFrameCount . . . 1705
76.15.82 ExceptFrames . . . 1705
76.15.83 ExceptionErrorMessage . . . 1706
76.15.84 ExceptObject . . . 1706
76.15.85 ExcludeLeadingPathDelimiter . . . 1706
76.15.86 ExcludeTrailingBackslash . . . 1707
76.15.87 ExcludeTrailingPathDelimiter . . . 1707
76.15.88 ExecuteProcess . . . 1707
76.15.89 ExeSearch . . . 1708
76.15.90 ExpandFileName . . . 1708
76.15.91 ExpandFileNameCase . . . 1709
76.15.92 ExpandUNCFileName . . . 1710
76.15.93 ExtractFileDir . . . 1710
76.15.94 ExtractFileDrive . . . 1711
76.15.95 ExtractFileExt . . . 1711
76.15.96 ExtractFileName . . . 1712
76.15.97 ExtractFilePath . . . 1712
76.15.98 ExtractRelativePath . . . 1712
76.15.99 ExtractShortPathName . . . 1713
76.15.100FileAge . . . 1714
76.15.101FileAgeUTC . . . 1714
76.15.102FileClose . . . 1715
76.15.103FileCreate . . . 1715
76.15.104FileDateToDateTime . . . 1716
76.15.105FileDateToUniversal . . . 1717
76.15.106FileExists . . . 1717
76.15.107FileFlush . . . 1718
76.15.108FileGetAttr . . . 1718
76.15.109FileGetDate . . . 1719
76.15.110FileGetDateTimeInfo . . . 1720
76.15.111FileGetDateUTC . . . 1720
76.15.112FileGetSymLinkTarget . . . 1720
76.15.113FileIsReadOnly . . . 1720
76.15.114FileOpen . . . 1721
76.15.115FileRead . . . 1722
76.15.116FileSearch . . . 1722

76.15.117FileSeek 1723
76.15.118FileSetAttr 1724
76.15.119FileSetDate 1725
76.15.120FileSetDateUTC 1725
76.15.121FileTruncate 1725
76.15.122FileWrite 1726
76.15.123FindClose 1726
76.15.124FindCmdLineSwitch 1726
76.15.125FindFirst 1727
76.15.126FindNext 1728
76.15.127FloattoCurr 1728
76.15.128FloatToDateTime 1729
76.15.129FloatToDecimal 1729
76.15.130FloatToStr 1729
76.15.131FloatToStrF 1731
76.15.132FloatToText 1732
76.15.133FloatToTextFmt 1734
76.15.134FmtStr 1734
76.15.135ForceDirectories 1735
76.15.136Format 1735
76.15.137FormatBuf 1742
76.15.138FormatCurr 1742
76.15.139FormatDateTime 1743
76.15.140FormatFloat 1743
76.15.141FreeAndNil 1745
76.15.142FreeMemAndNil 1745
76.15.143GetAppConfigDir 1746
76.15.144GetAppConfigFile 1746
76.15.145GetCompiledArchitecture 1747
76.15.146GetCompiledPlatform 1747
76.15.147GetCurrentDir 1747
76.15.148GetDirs 1748
76.15.149GetDriveIDFromLetter 1748
76.15.150GetEnvironmentString 1749
76.15.151GetEnvironmentVariable 1749
76.15.152GetEnvironmentVariableCount 1749
76.15.153GetFileAsString 1750
76.15.154GetFileContents 1750
76.15.155GetFileHandle 1750
76.15.156GetLastOSError 1751

76.15.157GetLocalTime . . . 1751
76.15.158GetLocalTimeOffset . . . 1751
76.15.159GetModuleName . . . 1752
76.15.160GetTempDir . . . 1752
76.15.161GetTempFileName . . . 1752
76.15.162GetTickCount . . . 1753
76.15.163GetTickCount64 . . . 1753
76.15.164GetUniversalTime . . . 1753
76.15.165GetUserDir . . . 1753
76.15.166GuidCase . . . 1754
76.15.167GUIDToString . . . 1754
76.15.168HashName . . . 1754
76.15.169HookSignal . . . 1755
76.15.170IncAMonth . . . 1755
76.15.171IncludeLeadingPathDelimiter . . . 1755
76.15.172IncludeTrailingBackslash . . . 1756
76.15.173IncludeTrailingPathDelimiter . . . 1756
76.15.174IncMonth . . . 1756
76.15.175InquireSignal . . . 1757
76.15.176IntToHex . . . 1757
76.15.177IntToStr . . . 1758
76.15.178IsDelimiter . . . 1759
76.15.179IsEqualGUID . . . 1759
76.15.180IsFileNameCasePreserving . . . 1759
76.15.181IsFileNameCaseSensitive . . . 1759
76.15.182IsLeadChar . . . 1760
76.15.183IsLeapYear . . . 1760
76.15.184IsPathDelimiter . . . 1761
76.15.185IsValidIdent . . . 1761
76.15.186LastDelimiter . . . 1762
76.15.187LeftStr . . . 1762
76.15.188ListIndexError . . . 1762
76.15.189LoadStr . . . 1763
76.15.190LocalTimeToUniversal . . . 1763
76.15.191LowerCase . . . 1763
76.15.192MSecsToTimeStamp . . . 1764
76.15.193NewStr . . . 1764
76.15.194Now . . . 1765
76.15.195NowUTC . . . 1765
76.15.196OutOfMemoryError . . . 1765

76.15.197QuotedStr 1765
76.15.198RaiseLastOSError 1766
76.15.199RemoveDir 1766
76.15.200RenameFile 1767
76.15.201ReplaceDate 1767
76.15.202ReplaceTime 1768
76.15.203RightStr 1768
76.15.204SafeFormat 1768
76.15.205SafeLoadLibrary 1768
76.15.206SameFileName 1769
76.15.207SameStr 1769
76.15.208SameText 1769
76.15.209SetCurrentDir 1770
76.15.210SetDirSeparators 1770
76.15.211ShowException 1770
76.15.212Sleep 1771
76.15.213SScanf 1771
76.15.214StrAlloc 1772
76.15.215StrBufSize 1772
76.15.216StrByteType 1773
76.15.217strcat 1773
76.15.218StrCharLength 1774
76.15.219strcomp 1774
76.15.220StrCopy 1774
76.15.221StrDispose 1775
76.15.222strecopy 1775
76.15.223strend 1776
76.15.224StrFmt 1777
76.15.225stricomp 1777
76.15.226StringOf 1778
76.15.227StringReplace 1778
76.15.228StringToGUID 1779
76.15.229strlcat 1779
76.15.230strlcomp 1780
76.15.231StrLCopy 1780
76.15.232StrLen 1781
76.15.233StrLFmt 1781
76.15.234strlicomp 1782
76.15.235strlower 1783
76.15.236StrMove 1783

76.15.237strnew . . . 1784
76.15.238StrNextChar . . . 1785
76.15.239StrPas . . . 1785
76.15.240StrPCopy . . . 1785
76.15.241StrPLCopy . . . 1785
76.15.242strpos . . . 1786
76.15.243strrscan . . . 1786
76.15.244StrScan . . . 1787
76.15.245StrToBool . . . 1787
76.15.246StrToBoolDef . . . 1787
76.15.247StrToCurr . . . 1788
76.15.248StrToCurrDef . . . 1788
76.15.249StrToDate . . . 1788
76.15.250StrToDateDef . . . 1789
76.15.251StrToDateTime . . . 1790
76.15.252StrToDateTimeDef . . . 1790
76.15.253StrToDWord . . . 1791
76.15.254StrToDWordDef . . . 1791
76.15.255StrToFloat . . . 1791
76.15.256StrToFloatDef . . . 1793
76.15.257StrToInt . . . 1793
76.15.258StrToInt64 . . . 1794
76.15.259StrToInt64Def . . . 1794
76.15.260StrToIntDef . . . 1794
76.15.261StrToQWord . . . 1795
76.15.262StrToQWordDef . . . 1795
76.15.263StrToTime . . . 1795
76.15.264StrToTimeDef . . . 1796
76.15.265StrToUInt . . . 1797
76.15.266StrToUInt64 . . . 1797
76.15.267StrToUInt64Def . . . 1797
76.15.268StrToUIntDef . . . 1798
76.15.269strupper . . . 1798
76.15.270Supports . . . 1798
76.15.271SysErrorMessage . . . 1799
76.15.272SystemTimeToDateTime . . . 1799
76.15.273TextToFloat . . . 1800
76.15.274Time . . . 1801
76.15.275TimeStampToDateTime . . . 1801
76.15.276TimeStampToMSecs . . . 1802

76.15.277TimeToStr . . . 1802
76.15.278Trim . . . 1803
76.15.279TrimLeft . . . 1803
76.15.280TrimRight . . . 1804
76.15.281TryEncodeDate . . . 1805
76.15.282TryEncodeTime . . . 1805
76.15.283TryFloatToCurr . . . 1805
76.15.284TryStringToGUID . . . 1806
76.15.285TryStrToBool . . . 1806
76.15.286TryStrToCurr . . . 1806
76.15.287TryStrToDate . . . 1807
76.15.288TryStrToDateTime . . . 1807
76.15.289TryStrToDWord . . . 1808
76.15.290TryStrToFloat . . . 1808
76.15.291TryStrToInt . . . 1808
76.15.292TryStrToInt64 . . . 1809
76.15.293TryStrToQWord . . . 1809
76.15.294TryStrToTime . . . 1809
76.15.295TryStrToUInt . . . 1810
76.15.296TryStrToUInt64 . . . 1810
76.15.297UIntToStr . . . 1810
76.15.298UnhookSignal . . . 1810
76.15.299UnicodeCompareStr . . . 1811
76.15.300UnicodeCompareText . . . 1811
76.15.301UnicodeFmtStr . . . 1811
76.15.302UnicodeFormat . . . 1812
76.15.303UnicodeFormatBuf . . . 1812
76.15.304UnicodeLowerCase . . . 1812
76.15.305UnicodeSameStr . . . 1813
76.15.306UnicodeSameText . . . 1813
76.15.307UnicodeStringReplace . . . 1813
76.15.308UnicodeUpperCase . . . 1814
76.15.309UniversalTimeToLocal . . . 1814
76.15.310UniversalToFileDate . . . 1814
76.15.311UpperCase . . . 1814
76.15.312VendorName . . . 1815
76.15.313WideBytesOf . . . 1815
76.15.314WideCompareStr . . . 1815
76.15.315WideCompareText . . . 1816
76.15.316WideFmtStr . . . 1816

76.15.317WideFormat . . . 1816
76.15.318WideFormatBuf . . . 1817
76.15.319WideLowerCase . . . 1817
76.15.320WideSameStr . . . 1817
76.15.321WideSameText . . . 1817
76.15.322WideStrAlloc . . . 1818
76.15.323WideStringOf . . . 1818
76.15.324WideStringReplace . . . 1818
76.15.325WideUpperCase . . . 1819
76.15.326WrapText . . . 1819
76.16 TDateTimeInfoRec . . . 1819
76.16.1 Property overview . . . 1820
76.16.2 TDateTimeInfoRec.CreationTime . . . 1820
76.16.3 TDateTimeInfoRec.LastAccessTime . . . 1820
76.16.4 TDateTimeInfoRec.TimeStamp . . . 1820
76.17 TOSVersion . . . 1820
76.17.1 Method overview . . . 1822
76.17.2 Property overview . . . 1822
76.17.3 TOSVersion.Check . . . 1822
76.17.4 TOSVersion.ToString . . . 1822
76.17.5 TOSVersion.Architecture . . . 1822
76.17.6 TOSVersion.Build . . . 1822
76.17.7 TOSVersion.Major . . . 1823
76.17.8 TOSVersion.Minor . . . 1823
76.17.9 TOSVersion.Name . . . 1823
76.17.10 TOSVersion.Platform . . . 1823
76.17.11 TOSVersion.ServicePackMajor . . . 1823
76.17.12 TOSVersion.ServicePackMinor . . . 1823
76.17.13 TOSVersion.PrettyName . . . 1824
76.17.14 TOSVersion.LibCVersionMajor . . . 1824
76.17.15 TOSVersion.LibCVersionMinor . . . 1824
76.18 TRawbyteSearchRec . . . 1824
76.18.1 Method overview . . . 1825
76.18.2 Property overview . . . 1825
76.18.3 TRawbyteSearchRec.IsDirectory . . . 1825
76.18.4 TRawbyteSearchRec.IsCurrentOrParentDir . . . 1825
76.18.5 TRawbyteSearchRec.TimeStamp . . . 1825
76.18.6 TRawbyteSearchRec.TimeStampUTC . . . 1825
76.19 TRawbyteSymLinkRec . . . 1826
76.19.1 Property overview . . . 1826

76.19.2 TRawbyteSymLinkRec.TimeStamp . . . 1826
76.20 TUnicodeSearchRec . . . 1826
76.20.1 Method overview . . . 1827
76.20.2 Property overview . . . 1827
76.20.3 TUnicodeSearchRec.IsDirectory . . . 1827
76.20.4 TUnicodeSearchRec.IsCurrentOrParentDir . . . 1827
76.20.5 TUnicodeSearchRec.TimeStamp . . . 1827
76.20.6 TUnicodeSearchRec.TimeStampUTC . . . 1828
76.21 TUnicodeSymLinkRec . . . 1828
76.21.1 Property overview . . . 1828
76.21.2 TUnicodeSymLinkRec.TimeStamp . . . 1829
76.22 EAbort . . . 1829
76.22.1 Description . . . 1829
76.23 EAbstractError . . . 1829
76.23.1 Description . . . 1829
76.24 EAccessViolation . . . 1829
76.24.1 Description . . . 1829
76.25 EArgumentException . . . 1829
76.25.1 Description . . . 1829
76.26 EArgumentNilException . . . 1829
76.26.1 Description . . . 1829
76.27 EArgumentOutOfRangeException . . . 1830
76.27.1 Description . . . 1830
76.28 EAssertionFailed . . . 1830
76.28.1 Description . . . 1830
76.29 EBusError . . . 1830
76.29.1 Description . . . 1830
76.30 ECFError . . . 1830
76.31 EControlC . . . 1830
76.31.1 Description . . . 1830
76.32 EConvertError . . . 1830
76.32.1 Description . . . 1830
76.33 EDirectoryNotFoundException . . . 1830
76.33.1 Description . . . 1830
76.34 EDivByZero . . . 1831
76.34.1 Description . . . 1831
76.35 EEncodingError . . . 1831
76.35.1 Description . . . 1831
76.36 EExternal . . . 1831
76.36.1 Description . . . 1831

76.37 EExternalException . . . 1831
76.37.1 Description . . . 1831
76.38 EFileNotFoundException . . . 1831
76.38.1 Description . . . 1831
76.39 EFormatError . . . 1832
76.39.1 Description . . . 1832
76.40 EHeapMemoryError . . . 1832
76.40.1 Description . . . 1832
76.40.2 Method overview . . . 1832
76.40.3 EHeapMemoryError.FreeInstance . . . 1832
76.41 EInOutArgumentException . . . 1832
76.41.1 Method overview . . . 1832
76.41.2 EInOutArgumentException.Create . . . 1832
76.41.3 EInOutArgumentException.CreateRes . . . 1832
76.41.4 EInOutArgumentException.CreateFmt . . . 1833
76.42 EInOutError . . . 1833
76.42.1 Description . . . 1833
76.43 EIntError . . . 1833
76.43.1 Description . . . 1833
76.44 EIntfCastError . . . 1833
76.44.1 Description . . . 1833
76.45 EIntOverflow . . . 1833
76.45.1 Description . . . 1833
76.46 EInvalidCast . . . 1833
76.46.1 Description . . . 1833
76.47 EInvalidContainer . . . 1834
76.47.1 Description . . . 1834
76.48 EInvalidInsert . . . 1834
76.48.1 Description . . . 1834
76.49 EInvalidOp . . . 1834
76.49.1 Description . . . 1834
76.50 EInvalidOpException . . . 1834
76.50.1 Description . . . 1834
76.51 EInvalidPointer . . . 1834
76.51.1 Description . . . 1834
76.52 EListError . . . 1834
76.53 EMathError . . . 1834
76.53.1 Description . . . 1834
76.54 EMonitor . . . 1835
76.55 EMonitorLockException . . . 1835

76.56 ENoConstructException . . . 1835
76.56.1 Description . . . 1835
76.57 ENoDynLibsSupport . . . 1835
76.57.1 Description . . . 1835
76.58 ENoMonitorSupportException . . . 1835
76.59 ENoThreadSupport . . . 1835
76.59.1 Description . . . 1835
76.60 ENotImplemented . . . 1835
76.60.1 Description . . . 1835
76.61 ENotSupportedException . . . 1835
76.61.1 Description . . . 1835
76.62 ENoWideStringSupport . . . 1836
76.62.1 Description . . . 1836
76.63 EObjectCheck . . . 1836
76.63.1 Description . . . 1836
76.64 EObjectDisposed . . . 1836
76.65 EOperationCancelled . . . 1836
76.66 EOSError . . . 1836
76.66.1 Description . . . 1836
76.67 EOutOfMemory . . . 1836
76.67.1 Description . . . 1836
76.68 EOverflow . . . 1836
76.68.1 Description . . . 1836
76.69 EPackageError . . . 1837
76.69.1 Description . . . 1837
76.70 EPathNotFoundException . . . 1837
76.70.1 Description . . . 1837
76.71 EPathTooLongException . . . 1837
76.71.1 Description . . . 1837
76.72 EPrivilege . . . 1837
76.72.1 Description . . . 1837
76.73 EProgrammerNotFound . . . 1837
76.73.1 Description . . . 1837
76.74 EPropReadOnly . . . 1837
76.74.1 Description . . . 1837
76.75 EPropWriteOnly . . . 1838
76.75.1 Description . . . 1838
76.76 ERangeError . . . 1838
76.76.1 Description . . . 1838
76.77 ESafecallException . . . 1838

76.77.1 Description . . . 1838
76.78 ESigQuit . . . 1838
76.78.1 Description . . . 1838
76.79 EStackOverflow . . . 1838
76.79.1 Description . . . 1838
76.80 EThreadError . . . 1838
76.81 EUnderflow . . . 1838
76.81.1 Description . . . 1838
76.82 EVariantError . . . 1839
76.82.1 Description . . . 1839
76.82.2 Method overview . . . 1839
76.82.3 EVariantError.CreateCode . . . 1839
76.83 Exception . . . 1839
76.83.1 Description . . . 1839
76.83.2 Method overview . . . 1840
76.83.3 Property overview . . . 1840
76.83.4 Exception.Create . . . 1840
76.83.5 Exception.CreateFmt . . . 1840
76.83.6 Exception.CreateRes . . . 1840
76.83.7 Exception.CreateResFmt . . . 1841
76.83.8 Exception.CreateHelp . . . 1841
76.83.9 Exception.CreateFmtHelp . . . 1841
76.83.10 Exception.CreateResHelp . . . 1841
76.83.11 Exception.CreateResFmtHelp . . . 1842
76.83.12 Exception.ToString . . . 1842
76.83.13 Exception.GetBaseException . . . 1842
76.83.14 Exception.HelpContext . . . 1842
76.83.15 Exception.Message . . . 1842
76.84 EZeroDivide . . . 1843
76.84.1 Description . . . 1843
76.85 IReadWriteSync . . . 1843
76.85.1 Description . . . 1843
76.85.2 Method overview . . . 1843
76.85.3 IReadWriteSync.BeginRead . . . 1843
76.85.4 IReadWriteSync.EndRead . . . 1843
76.85.5 IReadWriteSync.BeginWrite . . . 1844
76.85.6 IReadWriteSync.EndWrite . . . 1844
76.86 TANSISTRINGBUILDER . . . 1844
76.86.1 Description . . . 1844
76.86.2 Method overview . . . 1845

76.86.3 Property overview . . . 1845
76.86.4 TANSISTRINGBUILDER.Create . . . 1845
76.86.5 TANSISTRINGBUILDER.Append . . . 1845
76.86.6 TANSISTRINGBUILDER.AppendFormat . . . 1846
76.86.7 TANSISTRINGBUILDER.AppendLine . . . 1847
76.86.8 TANSISTRINGBUILDER.Clear . . . 1847
76.86.9 TANSISTRINGBUILDER.CopyTo . . . 1847
76.86.10 TANSISTRINGBUILDER.EnsureCapacity . . . 1847
76.86.11 TANSISTRINGBUILDER.Equals . . . 1848
76.86.12 TANSISTRINGBUILDER.Insert . . . 1848
76.86.13 TANSISTRINGBUILDER.Remove . . . 1849
76.86.14 TANSISTRINGBUILDER.Replace . . . 1849
76.86.15 TANSISTRINGBUILDER.ToString . . . 1850
76.86.16 TANSISTRINGBUILDER.Chars . . . 1850
76.86.17 TANSISTRINGBUILDER.Length . . . 1850
76.86.18 TANSISTRINGBUILDER.Capacity . . . 1850
76.86.19 TANSISTRINGBUILDER.MaxCapacity . . . 1851
76.87 TBigEndianUnicodeEncoding . . . 1851
76.87.1 Description . . . 1851
76.87.2 Method overview . . . 1851
76.87.3 TBigEndianUnicodeEncoding.Clone . . . 1851
76.87.4 TBigEndianUnicodeEncoding.GetPreamble . . . 1851
76.88 TBooleanHelper . . . 1852
76.88.1 Description . . . 1852
76.88.2 Method overview . . . 1852
76.88.3 TBooleanHelper.Parse . . . 1852
76.88.4 TBooleanHelper.Size . . . 1852
76.88.5 TBooleanHelper.ToString . . . 1852
76.88.6 TBooleanHelper.TryToParse . . . 1853
76.88.7 TBooleanHelper.ToInteger . . . 1853
76.89 TByteBoolHelper . . . 1853
76.89.1 Description . . . 1853
76.89.2 Method overview . . . 1854
76.89.3 TByteBoolHelper.Parse . . . 1854
76.89.4 TByteBoolHelper.Size . . . 1854
76.89.5 TByteBoolHelper.ToString . . . 1854
76.89.6 TByteBoolHelper.TryToParse . . . 1855
76.89.7 TByteBoolHelper.ToInteger . . . 1855
76.90 TByteHelper . . . 1855
76.90.1 Description . . . 1855

76.90.2 Method overview . . . 1855
76.90.3 TByteHelper.Parse . . . 1856
76.90.4 TByteHelper.Size . . . 1856
76.90.5 TByteHelper.ToString . . . 1856
76.90.6 TByteHelper.TryParse . . . 1856
76.90.7 TByteHelper.ToBoolean . . . 1857
76.90.8 TByteHelper.ToDouble . . . 1857
76.90.9 TByteHelper.ToExtended . . . 1857
76.90.10 TByteHelper.ToBinString . . . 1857
76.90.11 TByteHelper.ToHexString . . . 1858
76.90.12 TByteHelper.ToSingle . . . 1858
76.90.13 TByteHelper.SetBit . . . 1858
76.90.14 TByteHelper.ClearBit . . . 1858
76.90.15 TByteHelper.ToggleBit . . . 1859
76.90.16 TByteHelper.TestBit . . . 1859
76.91 TCardinalHelper . . . 1859
76.91.1 Description . . . 1859
76.91.2 Method overview . . . 1860
76.91.3 TCardinalHelper.Parse . . . 1860
76.91.4 TCardinalHelper.Size . . . 1860
76.91.5 TCardinalHelper.ToString . . . 1860
76.91.6 TCardinalHelper.TryParse . . . 1861
76.91.7 TCardinalHelper.ToBoolean . . . 1861
76.91.8 TCardinalHelper.ToDouble . . . 1861
76.91.9 TCardinalHelper.ToExtended . . . 1861
76.91.10 TCardinalHelper.ToBinString . . . 1862
76.91.11 TCardinalHelper.ToHexString . . . 1862
76.91.12 TCardinalHelper.ToSingle . . . 1862
76.91.13 TCardinalHelper.SetBit . . . 1862
76.91.14 TCardinalHelper.ClearBit . . . 1863
76.91.15 TCardinalHelper.ToggleBit . . . 1863
76.91.16 TCardinalHelper.TestBit . . . 1863
76.92 TDoubleHelper . . . 1863
76.92.1 Description . . . 1863
76.92.2 Method overview . . . 1864
76.92.3 Property overview . . . 1864
76.92.4 TDoubleHelper.IsInfinity . . . 1864
76.92.5 TDoubleHelper.IsNan . . . 1864
76.92.6 TDoubleHelper.IsNegativeInfinity . . . 1865
76.92.7 TDoubleHelper.IsPositiveInfinity . . . 1865

76.92.8 TDoubleHelper.Parse . . . 1865
76.92.9 TDoubleHelper.Size . . . 1866
76.92.10 TDoubleHelper.ToString . . . 1866
76.92.11 TDoubleHelper.TryParse . . . 1867
76.92.12 TDoubleHelper.BuildUp . . . 1867
76.92.13 TDoubleHelper.Exponent . . . 1867
76.92.14 TDoubleHelper.Fraction . . . 1867
76.92.15 TDoubleHelper.Mantissa . . . 1868
76.92.16 TDoubleHelper.SpecialType . . . 1868
76.92.17 TDoubleHelper.Bytes . . . 1868
76.92.18 TDoubleHelper.Words . . . 1868
76.92.19 TDoubleHelper.Sign . . . 1869
76.92.20 TDoubleHelper.Exp . . . 1869
76.92.21 TDoubleHelper.Frac . . . 1869
76.93 TEncoding . . . 1869
76.93.1 Description . . . 1869
76.93.2 Method overview . . . 1870
76.93.3 Property overview . . . 1870
76.93.4 TEncoding.Clone . . . 1870
76.93.5 TEncoding.Convert . . . 1870
76.93.6 TEncoding.IsStandardEncoding . . . 1871
76.93.7 TEncoding.GetBufferEncoding . . . 1871
76.93.8 TEncoding.GetEncoding . . . 1872
76.93.9 TEncoding.GetMaxByteCount . . . 1872
76.93.10 TEncoding.GetMaxCharCount . . . 1872
76.93.11 TEncoding.GetPreamble . . . 1872
76.93.12 TEncoding.GetString . . . 1873
76.93.13 TEncoding.CodePage . . . 1873
76.93.14 TEncoding.EncodingName . . . 1873
76.93.15 TEncoding.IsSingleByte . . . 1874
76.93.16 TEncoding.ANSI . . . 1874
76.93.17 TEncoding.ASCII . . . 1874
76.93.18 TEncoding.BigEndianUnicode . . . 1874
76.93.19 TEncoding.Default . . . 1875
76.93.20 TEncoding.SystemEncoding . . . 1875
76.93.21 TEncoding.Unicode . . . 1875
76.93.22 TEncoding.UTF7 . . . 1875
76.93.23 TEncoding.UTF8 . . . 1876
76.94 TExtendedHelper . . . 1876
76.94.1 Description . . . 1876

76.94.2 Method overview . . . 1876
76.94.3 Property overview . . . 1877
76.94.4 TExtendedHelper.ToString . . . 1877
76.94.5 TExtendedHelper.Parse . . . 1877
76.94.6 TExtendedHelper.TryParse . . . 1878
76.94.7 TExtendedHelper.IsNan . . . 1878
76.94.8 TExtendedHelper.IsInfinity . . . 1878
76.94.9 TExtendedHelper.IsNegativeInfinity . . . 1879
76.94.10 TExtendedHelper.IsPositiveInfinity . . . 1879
76.94.11 TExtendedHelper.Size . . . 1879
76.94.12 TExtendedHelper.BuildUp . . . 1880
76.94.13 TExtendedHelper.Exponent . . . 1880
76.94.14 TExtendedHelper.Fraction . . . 1880
76.94.15 TExtendedHelper.Mantissa . . . 1880
76.94.16 TExtendedHelper.SpecialType . . . 1881
76.94.17 TExtendedHelper.Bytes . . . 1881
76.94.18 TExtendedHelper.Words . . . 1881
76.94.19 TExtendedHelper.Sign . . . 1881
76.94.20 TExtendedHelper.Exp . . . 1882
76.94.21 TExtendedHelper.Frac . . . 1882
76.95 TGuidHelper . . . 1882
76.95.1 Description . . . 1882
76.95.2 Method overview . . . 1882
76.95.3 TGuidHelper.Create . . . 1882
76.95.4 TGuidHelper.NewGuid . . . 1883
76.95.5 TGuidHelper.ToByteArray . . . 1883
76.95.6 TGuidHelper.ToString . . . 1884
76.96 TInt64Helper . . . 1884
76.96.1 Description . . . 1884
76.96.2 Method overview . . . 1884
76.96.3 TInt64Helper.Parse . . . 1884
76.96.4 TInt64Helper.Size . . . 1885
76.96.5 TInt64Helper.ToString . . . 1885
76.96.6 TInt64Helper.TryParse . . . 1885
76.96.7 TInt64Helper.ToBoolean . . . 1885
76.96.8 TInt64Helper.ToDouble . . . 1886
76.96.9 TInt64Helper.ToExtended . . . 1886
76.96.10 TInt64Helper.ToBinString . . . 1886
76.96.11 TInt64Helper.ToHexString . . . 1886
76.96.12 TInt64Helper.ToSingle . . . 1887

76.96.13 TInt64Helper.SetBit . . . 1887
76.96.14 TInt64Helper.ClearBit . . . 1887
76.96.15 TInt64Helper.ToggleBit . . . 1887
76.96.16 TInt64Helper.TestBit . . . 1888
76.97 TIntegerHelper . . . 1888
76.97.1 Description . . . 1888
76.97.2 Method overview . . . 1888
76.97.3 TIntegerHelper.Size . . . 1888
76.97.4 TIntegerHelper.ToString . . . 1889
76.97.5 TIntegerHelper.Parse . . . 1889
76.97.6 TIntegerHelper.TryParse . . . 1889
76.97.7 TIntegerHelper.ToBoolean . . . 1889
76.97.8 TIntegerHelper.ToDouble . . . 1890
76.97.9 TIntegerHelper.ToExtended . . . 1890
76.97.10 TIntegerHelper.ToBinString . . . 1890
76.97.11 TIntegerHelper.ToHexString . . . 1890
76.97.12 TIntegerHelper.ToSingle . . . 1891
76.97.13 TIntegerHelper.SetBit . . . 1891
76.97.14 TIntegerHelper.ClearBit . . . 1891
76.97.15 TIntegerHelper.ToggleBit . . . 1891
76.97.16 TIntegerHelper.TestBit . . . 1892
76.98 TLongBoolHelper . . . 1892
76.98.1 Description . . . 1892
76.98.2 Method overview . . . 1892
76.98.3 TLongBoolHelper.Parse . . . 1892
76.98.4 TLongBoolHelper.Size . . . 1893
76.98.5 TLongBoolHelper.ToString . . . 1893
76.98.6 TLongBoolHelper.TryToParse . . . 1893
76.98.7 TLongBoolHelper.ToInteger . . . 1893
76.99 TMBCSEncoding . . . 1894
76.99.1 Description . . . 1894
76.99.2 Method overview . . . 1894
76.99.3 TMBCSEncoding.Create . . . 1894
76.99.4 TMBCSEncoding.Clone . . . 1894
76.99.5 TMBCSEncoding.GetMaxByteCount . . . 1894
76.99.6 TMBCSEncoding.GetMaxCharCount . . . 1895
76.99.7 TMBCSEncoding.GetPreamble . . . 1895
76.100TMREWException . . . 1895
76.100.1 Description . . . 1895
76.101TMultiReadExclusiveWriteSynchronizer . . . 1895

76.101.1 Description . . . 1895
76.101.2 Interfaces overview . . . 1896
76.101.3 Method overview . . . 1896
76.101.4 Property overview . . . 1896
76.101.5 TMultiReadExclusiveWriteSynchronizer.Create . . . 1896
76.101.6 TMultiReadExclusiveWriteSynchronizer.Destroy . . . 1896
76.101.7 TMultiReadExclusiveWriteSynchronizer.Beginwrite . . . 1897
76.101.8 TMultiReadExclusiveWriteSynchronizer.Endwrite . . . 1897
76.101.9 TMultiReadExclusiveWriteSynchronizer.Beginread . . . 1897
76.101.10 TMultiReadExclusiveWriteSynchronizer.Endread . . . 1897
76.101.11 TMultiReadExclusiveWriteSynchronizer.RevisionLevel . . . 1898
76.101.12 TMultiReadExclusiveWriteSynchronizer.WriterThreadID . . . 1898
76.102 TNativeIntHelper . . . 1898
76.102.1 Description . . . 1898
76.102.2 Method overview . . . 1898
76.102.3 TNativeIntHelper.Parse . . . 1898
76.102.4 TNativeIntHelper.Size . . . 1899
76.102.5 TNativeIntHelper.ToString . . . 1899
76.102.6 TNativeIntHelper.TryParse . . . 1899
76.102.7 TNativeIntHelper.ToBoolean . . . 1900
76.102.8 TNativeIntHelper.ToDouble . . . 1900
76.102.9 TNativeIntHelper.ToExtended . . . 1900
76.102.10 TNativeIntHelper.ToBinString . . . 1900
76.102.11 TNativeIntHelper.ToHexString . . . 1901
76.102.12 TNativeIntHelper.ToSingle . . . 1901
76.102.13 TNativeIntHelper.SetBit . . . 1901
76.102.14 TNativeIntHelper.ClearBit . . . 1901
76.102.15 TNativeIntHelper.ToggleBit . . . 1902
76.102.16 TNativeIntHelper.TestBit . . . 1902
76.103 TNativeUIntHelper . . . 1902
76.103.1 Description . . . 1902
76.103.2 Method overview . . . 1903
76.103.3 TNativeUIntHelper.Parse . . . 1903
76.103.4 TNativeUIntHelper.Size . . . 1903
76.103.5 TNativeUIntHelper.ToString . . . 1903
76.103.6 TNativeUIntHelper.TryParse . . . 1904
76.103.7 TNativeUIntHelper.ToBoolean . . . 1904
76.103.8 TNativeUIntHelper.ToDouble . . . 1904
76.103.9 TNativeUIntHelper.ToExtended . . . 1904
76.103.10 TNativeUIntHelper.ToBinString . . . 1905

76.103.11TNativeUIntHelper.ToHexString . . . 1905
76.103.12TNativeUIntHelper.ToSingle . . . 1905
76.103.13TNativeUIntHelper.SetBit . . . 1905
76.103.14TNativeUIntHelper.ClearBit . . . 1906
76.103.15TNativeUIntHelper.ToggleBit . . . 1906
76.103.16TNativeUIntHelper.TestBit . . . 1906
76.104TQWordHelper . . . 1906
76.104.1 Description . . . 1906
76.104.2 Method overview . . . 1907
76.104.3 TQWordHelper.Parse . . . 1907
76.104.4 TQWordHelper.Size . . . 1907
76.104.5 TQWordHelper.ToString . . . 1907
76.104.6 TQWordHelper.TryParse . . . 1908
76.104.7 TQWordHelper.ToBoolean . . . 1908
76.104.8 TQWordHelper.ToDouble . . . 1908
76.104.9 TQWordHelper.ToExtended . . . 1908
76.104.10TQWordHelper.ToBinString . . . 1909
76.104.11TQWordHelper.ToHexString . . . 1909
76.104.12TQWordHelper.ToSingle . . . 1909
76.104.13TQWordHelper.SetBit . . . 1909
76.104.14TQWordHelper.ClearBit . . . 1910
76.104.15TQWordHelper.ToggleBit . . . 1910
76.104.16TQWordHelper.TestBit . . . 1910
76.105TShortIntHelper . . . 1910
76.105.1 Description . . . 1910
76.105.2 Method overview . . . 1911
76.105.3 TShortIntHelper.Parse . . . 1911
76.105.4 TShortIntHelper.Size . . . 1911
76.105.5 TShortIntHelper.ToString . . . 1911
76.105.6 TShortIntHelper.TryParse . . . 1912
76.105.7 TShortIntHelper.ToBoolean . . . 1912
76.105.8 TShortIntHelper.ToDouble . . . 1912
76.105.9 TShortIntHelper.ToExtended . . . 1912
76.105.10TShortIntHelper.ToBinString . . . 1913
76.105.11TShortIntHelper.ToHexString . . . 1913
76.105.12TShortIntHelper.ToSingle . . . 1913
76.105.13TShortIntHelper.SetBit . . . 1913
76.105.14TShortIntHelper.ClearBit . . . 1914
76.105.15TShortIntHelper.ToggleBit . . . 1914
76.105.16TShortIntHelper.TestBit . . . 1914

76.106TSimpleRWSync 1914
76.106.1 Description 1914
76.106.2 Interfaces overview 1915
76.106.3 Method overview 1915
76.106.4 TSimpleRWSync.Create 1915
76.106.5 TSimpleRWSync.Destroy 1915
76.106.6 TSimpleRWSync.Beginwrite 1915
76.106.7 TSimpleRWSync.Endwrite 1916
76.106.8 TSimpleRWSync.Beginread 1916
76.106.9 TSimpleRWSync.Endread 1916
76.107TSingleHelper 1916
76.107.1 Description 1916
76.107.2 Method overview 1917
76.107.3 Property overview 1917
76.107.4 TSingleHelper.IsNan 1917
76.107.5 TSingleHelper.IsInfinity 1917
76.107.6 TSingleHelper.IsNegativeInfinity 1918
76.107.7 TSingleHelper.IsPositiveInfinity 1918
76.107.8 TSingleHelper.Parse 1918
76.107.9 TSingleHelper.Size 1919
76.107.10TSingleHelper.ToString 1919
76.107.11TSingleHelper.TryParse 1920
76.107.12TSingleHelper.BuildUp 1920
76.107.13TSingleHelper.Exponent 1920
76.107.14TSingleHelper.Fraction 1920
76.107.15TSingleHelper.Mantissa 1921
76.107.16TSingleHelper.SpecialType 1921
76.107.17TSingleHelper.Bytes 1921
76.107.18TSingleHelper.Words 1921
76.107.19TSingleHelper.Sign 1922
76.107.20TSingleHelper.Exp 1922
76.107.21TSingleHelper.Frac 1922
76.108TSmallIntHelper 1922
76.108.1 Description 1922
76.108.2 Method overview 1923
76.108.3 TSmallIntHelper.Parse 1923
76.108.4 TSmallIntHelper.Size 1923
76.108.5 TSmallIntHelper.ToString 1923
76.108.6 TSmallIntHelper.TryParse 1924
76.108.7 TSmallIntHelper.ToBoolean 1924

76.108.8 TSmallIntHelper.ToBinString . . . 1924
76.108.9 TSmallIntHelper.ToHexString . . . 1924
76.108.10TSmallIntHelper.ToSingle . . . 1925
76.108.11TSmallIntHelper.ToDouble . . . 1925
76.108.12TSmallIntHelper.ToExtended . . . 1925
76.108.13TSmallIntHelper.SetBit . . . 1925
76.108.14TSmallIntHelper.ClearBit . . . 1926
76.108.15TSmallIntHelper.ToggleBit . . . 1926
76.108.16TSmallIntHelper.TestBit . . . 1926
76.109TStringHelper . . . 1926
76.109.1 Description . . . 1926
76.109.2 Method overview . . . 1928
76.109.3 Property overview . . . 1929
76.109.4 TStringHelper.Compare . . . 1929
76.109.5 TStringHelper.CompareOrdinal . . . 1930
76.109.6 TStringHelper.CompareText . . . 1930
76.109.7 TStringHelper.Copy . . . 1930
76.109.8 TStringHelper.Create . . . 1931
76.109.9 TStringHelper.EndsText . . . 1931
76.109.10TStringHelper.Equals . . . 1931
76.109.11TStringHelper.Format . . . 1932
76.109.12TStringHelper.IsNullOrEmpty . . . 1932
76.109.13TStringHelper.IsNullOrWhiteSpace . . . 1932
76.109.14TStringHelper.Join . . . 1932
76.109.15TStringHelper.LowerCase . . . 1933
76.109.16TStringHelper.Parse . . . 1933
76.109.17TStringHelper.ToBoolean . . . 1933
76.109.18TStringHelper.ToDouble . . . 1934
76.109.19TStringHelper.ToExtended . . . 1934
76.109.20TStringHelper.ToInt64 . . . 1934
76.109.21TStringHelper.ToInteger . . . 1935
76.109.22TStringHelper.ToSingle . . . 1935
76.109.23TStringHelper.UpperCase . . . 1935
76.109.24TStringHelper.CompareTo . . . 1936
76.109.25TStringHelper.Contains . . . 1936
76.109.26TStringHelper.CopyTo . . . 1936
76.109.27TStringHelper.CountChar . . . 1936
76.109.28TStringHelper.DeQuotedString . . . 1937
76.109.29TStringHelper.EndsWith . . . 1937
76.109.30TStringHelper.GetHashCode . . . 1937

76.109.31TStringHelper.IndexOf . . . 1937
76.109.32TStringHelper.IndexOfUnQuoted . . . 1938
76.109.33TStringHelper.IndexOfAny . . . 1938
76.109.34TStringHelper.IndexOfAnyUnquoted . . . 1939
76.109.35TStringHelper.Insert . . . 1939
76.109.36TStringHelper.IsDelimiter . . . 1940
76.109.37TStringHelper.IsEmpty . . . 1940
76.109.38TStringHelper.LastDelimiter . . . 1940
76.109.39TStringHelper.LastIndexOf . . . 1940
76.109.40TStringHelper.LastIndexOfAny . . . 1941
76.109.41TStringHelper.PadLeft . . . 1941
76.109.42TStringHelper.PadRight . . . 1942
76.109.43TStringHelper.QuotedString . . . 1942
76.109.44TStringHelper.Remove . . . 1942
76.109.45TStringHelper.Replace . . . 1943
76.109.46TStringHelper.Split . . . 1943
76.109.47TStringHelper.StartsWith . . . 1944
76.109.48TStringHelper.Substring . . . 1944
76.109.49TStringHelper.ToCharArray . . . 1945
76.109.50TStringHelper.ToLower . . . 1945
76.109.51TStringHelper.ToLowerInvariant . . . 1945
76.109.52TStringHelper.ToUpper . . . 1946
76.109.53TStringHelper.ToUpperInvariant . . . 1946
76.109.54TStringHelper.Trim . . . 1946
76.109.55TStringHelper.TrimLeft . . . 1946
76.109.56TStringHelper.TrimRight . . . 1947
76.109.57TStringHelper.TrimEnd . . . 1947
76.109.58TStringHelper.TrimStart . . . 1947
76.109.59TStringHelper.Chars . . . 1947
76.109.60TStringHelper.Length . . . 1948
76.110TUnicodeEncoding . . . 1948
76.110.1 Description . . . 1948
76.110.2 Method overview . . . 1948
76.110.3 TUnicodeEncoding.Create . . . 1948
76.110.4 TUnicodeEncoding.Clone . . . 1948
76.110.5 TUnicodeEncoding.GetMaxByteCount . . . 1949
76.110.6 TUnicodeEncoding.GetMaxCharCount . . . 1949
76.110.7 TUnicodeEncoding.GetPreamble . . . 1949
76.111TUNICODESTRINGBUILDER . . . 1949
76.111.1 Description . . . 1949

76.111.2 Method overview . . . 1950
76.111.3 Property overview . . . 1950
76.111.4 TUNICODESTRINGBUILDER.Create . . . 1950
76.111.5 TUNICODESTRINGBUILDER.Append . . . 1950
76.111.6 TUNICODESTRINGBUILDER.AppendFormat . . . 1951
76.111.7 TUNICODESTRINGBUILDER.AppendLine . . . 1952
76.111.8 TUNICODESTRINGBUILDER.Clear . . . 1952
76.111.9 TUNICODESTRINGBUILDER.CopyTo . . . 1952
76.111.10TUNICODESTRINGBUILDER.EnsureCapacity . . . 1952
76.111.11TUNICODESTRINGBUILDER.Equals . . . 1953
76.111.12TUNICODESTRINGBUILDER.Insert . . . 1953
76.111.13TUNICODESTRINGBUILDER.Remove . . . 1954
76.111.14TUNICODESTRINGBUILDER.Replace . . . 1954
76.111.15TUNICODESTRINGBUILDER.ToString . . . 1955
76.111.16TUNICODESTRINGBUILDER.Chars . . . 1955
76.111.17TUNICODESTRINGBUILDER.Length . . . 1955
76.111.18TUNICODESTRINGBUILDER.Capacity . . . 1955
76.111.19TUNICODESTRINGBUILDER.MaxCapacity . . . 1956
76.112TUTF7Encoding . . . 1956
76.112.1 Description . . . 1956
76.112.2 Method overview . . . 1956
76.112.3 TUTF7Encoding.Create . . . 1956
76.112.4 TUTF7Encoding.Clone . . . 1956
76.112.5 TUTF7Encoding.GetMaxByteCount . . . 1957
76.112.6 TUTF7Encoding.GetMaxCharCount . . . 1957
76.113TUTF8Encoding . . . 1957
76.113.1 Description . . . 1957
76.113.2 Method overview . . . 1957
76.113.3 TUTF8Encoding.Create . . . 1957
76.113.4 TUTF8Encoding.Clone . . . 1958
76.113.5 TUTF8Encoding.GetMaxByteCount . . . 1958
76.113.6 TUTF8Encoding.GetMaxCharCount . . . 1958
76.113.7 TUTF8Encoding.GetPreamble . . . 1958
76.114TWordBoolHelper . . . 1959
76.114.1 Description . . . 1959
76.114.2 Method overview . . . 1959
76.114.3 TWordBoolHelper.Parse . . . 1959
76.114.4 TWordBoolHelper.Size . . . 1959
76.114.5 TWordBoolHelper.ToString . . . 1959
76.114.6 TWordBoolHelper.TryToParse . . . 1960

76.114.7 TWordBoolHelper.ToInteger . . . 1960
76.115TWordHelper . . . 1960
76.115.1 Description . . . 1960
76.115.2 Method overview . . . 1961
76.115.3 TWordHelper.Parse . . . 1961
76.115.4 TWordHelper.Size . . . 1961
76.115.5 TWordHelper.ToString . . . 1961
76.115.6 TWordHelper.TryParse . . . 1962
76.115.7 TWordHelper.ToBoolean . . . 1962
76.115.8 TWordHelper.ToDouble . . . 1962
76.115.9 TWordHelper.ToExtended . . . 1962
76.115.10TWordHelper.ToBinString . . . 1963
76.115.11TWordHelper.ToHexString . . . 1963
76.115.12TWordHelper.ToSingle . . . 1963
76.115.13TWordHelper.SetBit . . . 1963
76.115.14TWordHelper.ClearBit . . . 1964
76.115.15TWordHelper.ToggleBit . . . 1964
76.115.16TWordHelper.TestBit . . . 1964

77 Reference for unit 'Types' 1965
77.1 Used units . . . 1965
77.2 Overview . . . 1965
77.3 Constants, types and variables . . . 1965
77.3.1 Constants . . . 1965
77.3.2 Types . . . 1970
77.4 Procedures and functions . . . 1977
77.4.1 Bounds . . . 1977
77.4.2 CenteredRect . . . 1977
77.4.3 CenterPoint . . . 1977
77.4.4 EqualRect . . . 1978
77.4.5 InflateRect . . . 1978
77.4.6 IntersectRect . . . 1978
77.4.7 IntersectRectF . . . 1978
77.4.8 IsRectEmpty . . . 1979
77.4.9 MinPoint . . . 1979
77.4.10 MultiplyRect . . . 1979
77.4.11 NormalizeRect . . . 1979
77.4.12 NormalizeRectF . . . 1979
77.4.13 OffsetRect . . . 1979
77.4.14 Point . . . 1980

77.4.15 PointF . . . 1980
77.4.16 PtInRect . . . 1980
77.4.17 Rect . . . 1980
77.4.18 RectCenter . . . 1980
77.4.19 RectF . . . 1981
77.4.20 RectHeight . . . 1981
77.4.21 ScalePoint . . . 1981
77.4.22 Size . . . 1981
77.4.23 SplitRect . . . 1981
77.4.24 UnionRect . . . 1981
77.4.25 UnionRectF . . . 1982
77.5 TPoint . . . 1982
77.5.1 Method overview . . . 1983
77.5.2 TPoint.Zero . . . 1983
77.5.3 TPoint.Add . . . 1983
77.5.4 TPoint.Distance . . . 1983
77.5.5 TPoint.IsZero . . . 1984
77.5.6 TPoint.Subtract . . . 1984
77.5.7 TPoint.SetLocation . . . 1984
77.5.8 TPoint.Offset . . . 1984
77.5.9 TPoint.Angle . . . 1984
77.5.10 TPoint.PointInCircle . . . 1984
77.5.11 TPoint.equal(TPoint,TPoint):Boolean . . . 1984
77.5.12 TPoint.notequal(TPoint,TPoint):Boolean . . . 1984
77.5.13 TPoint.add(TPoint,TPoint):TPoint . . . 1985
77.5.14 TPoint.subtract(TPoint,TPoint):TPoint . . . 1985
77.5.15 TPoint.assign(TSmallPoint):TPoint . . . 1985
77.5.16 TPoint.explicit(TPoint):TSmallPoint . . . 1985
77.6 TPoint3D . . . 1985
77.6.1 Method overview . . . 1985
77.6.2 TPoint3D.Create . . . 1986
77.6.3 TPoint3D.Offset . . . 1986
77.7 TPointF . . . 1986
77.7.1 Method overview . . . 1988
77.7.2 TPointF.Add . . . 1988
77.7.3 TPointF.Distance . . . 1988
77.7.4 TPointF.DotProduct . . . 1989
77.7.5 TPointF.IsZero . . . 1989
77.7.6 TPointF.Subtract . . . 1989
77.7.7 TPointF.SetLocation . . . 1989

77.7.8 TPointF.Offset . . . 1989
77.7.9 TPointF.EqualsTo . . . 1989
77.7.10 TPointF.Scale . . . 1989
77.7.11 TPointF.Ceiling . . . 1989
77.7.12 TPointF.Truncate . . . 1990
77.7.13 TPointF.Floor . . . 1990
77.7.14 TPointF.Round . . . 1990
77.7.15 TPointF.Length . . . 1990
77.7.16 TPointF.Rotate . . . 1990
77.7.17 TPointF.Reflect . . . 1990
77.7.18 TPointF.MidPoint . . . 1990
77.7.19 TPointF.PointInCircle . . . 1990
77.7.20 TPointF.Zero . . . 1991
77.7.21 TPointF.Angle . . . 1991
77.7.22 TPointF.AngleCosine . . . 1991
77.7.23 TPointF.CrossProduct . . . 1991
77.7.24 TPointF.Normalize . . . 1991
77.7.25 TPointF.Create . . . 1991
77.7.26 TPointF.equal(TPointF,TPointF):Boolean . . . 1991
77.7.27 TPointF.notequal(TPointF,TPointF):Boolean . . . 1991
77.7.28 TPointF.add(TPointF,TPointF):TPointF . . . 1992
77.7.29 TPointF.subtract(TPointF,TPointF):TPointF . . . 1992
77.7.30 TPointF.negative(TPointF):TPointF . . . 1992
77.7.31 TPointF.multiply(TPointF,TPointF):TPointF . . . 1992
77.7.32 TPointF.multiply(TPointF,single):TPointF . . . 1992
77.7.33 TPointF.multiply(single,TPointF):TPointF . . . 1992
77.7.34 TPointF.divide(TPointF,single):TPointF . . . 1992
77.7.35 TPointF.assign(TPoint):TPointF . . . 1992
77.7.36 TPointF.power(TPointF,TPointF):Single . . . 1993
77.8 TRect . . . 1993
77.8.1 Method overview . . . 1994
77.8.2 Property overview . . . 1994
77.8.3 TRect.Create . . . 1995
77.8.4 TRect.equal(TRect,TRect):Boolean . . . 1995
77.8.5 TRect.notequal(TRect,TRect):Boolean . . . 1995
77.8.6 TRect.add(TRect,TRect):TRect . . . 1995
77.8.7 TRect.multiply(TRect,TRect):TRect . . . 1995
77.8.8 TRect.Empty . . . 1995
77.8.9 TRect.NormalizeRect . . . 1995
77.8.10 TRect.IsEmpty . . . 1995

77.8.11 TRect.Contains . . . 1996
77.8.12 TRect.IntersectsWith . . . 1996
77.8.13 TRect.Intersect . . . 1996
77.8.14 TRect.Union . . . 1996
77.8.15 TRect.Offset . . . 1996
77.8.16 TRect.SetLocation . . . 1996
77.8.17 TRect.Inflate . . . 1996
77.8.18 TRect.CenterPoint . . . 1996
77.8.19 TRect.SplitRect . . . 1997
77.8.20 TRect.Height . . . 1997
77.8.21 TRect.Width . . . 1997
77.8.22 TRect.Size . . . 1997
77.8.23 TRect.Location . . . 1997
77.9 TRectF . . . 1997
77.9.1 Method overview . . . 1999
77.9.2 Property overview . . . 1999
77.9.3 TRectF.Create . . . 1999
77.9.4 TRectF.equal(TRectF,TRectF):Boolean . . . 2000
77.9.5 TRectF.notequal(TRectF,TRectF):Boolean . . . 2000
77.9.6 TRectF.add(TRectF,TRectF):TRectF . . . 2000
77.9.7 TRectF.multiply(TRectF,TRectF):TRectF . . . 2000
77.9.8 TRectF.assign(TRect):TRectF . . . 2000
77.9.9 TRectF.Empty . . . 2000
77.9.10 TRectF.Intersect . . . 2000
77.9.11 TRectF.Union . . . 2001
77.9.12 TRectF.Ceiling . . . 2001
77.9.13 TRectF.CenterAt . . . 2001
77.9.14 TRectF.CenterPoint . . . 2001
77.9.15 TRectF.Contains . . . 2001
77.9.16 TRectF.EqualsTo . . . 2001
77.9.17 TRectF.Fit . . . 2001
77.9.18 TRectF.FitInto . . . 2001
77.9.19 TRectF.IntersectsWith . . . 2002
77.9.20 TRectF.IsEmpty . . . 2002
77.9.21 TRectF.PlaceInto . . . 2002
77.9.22 TRectF.Round . . . 2002
77.9.23 TRectF.SnapToPixel . . . 2002
77.9.24 TRectF.Truncate . . . 2002
77.9.25 TRectF.Inflate . . . 2002
77.9.26 TRectF.NormalizeRect . . . 2002

77.9.27 TRectF.Offset . . . 2003
77.9.28 TRectF.SetLocation . . . 2003
77.9.29 TRectF.Width . . . 2003
77.9.30 TRectF.Height . . . 2003
77.9.31 TRectF.Size . . . 2003
77.9.32 TRectF.Location . . . 2003
77.10 TSize . . . 2003
77.10.1 Method overview . . . 2004
77.10.2 Property overview . . . 2004
77.10.3 TSize.Add . . . 2004
77.10.4 TSize.Distance . . . 2004
77.10.5 TSize.IsZero . . . 2005
77.10.6 TSize.Subtract . . . 2005
77.10.7 TSize.equal(TSize,TSize):Boolean . . . 2005
77.10.8 TSize.notequal(TSize,TSize):Boolean . . . 2005
77.10.9 TSize.add(TSize,TSize):TSize . . . 2005
77.10.10 TSize.subtract(TSize,TSize):TSize . . . 2005
77.10.11 TSize.Width . . . 2005
77.10.12 TSize.Height . . . 2005
77.11 TSizeF . . . 2006
77.11.1 Method overview . . . 2007
77.11.2 Property overview . . . 2007
77.11.3 TSizeF.Add . . . 2007
77.11.4 TSizeF.Distance . . . 2007
77.11.5 TSizeF.IsZero . . . 2007
77.11.6 TSizeF.Subtract . . . 2008
77.11.7 TSizeF.SwapDimensions . . . 2008
77.11.8 TSizeF.Scale . . . 2008
77.11.9 TSizeF.Ceiling . . . 2008
77.11.10 TSizeF.Truncate . . . 2008
77.11.11 TSizeF.Floor . . . 2008
77.11.12 TSizeF.Round . . . 2008
77.11.13 TSizeF.Length . . . 2008
77.11.14 TSizeF.Create . . . 2009
77.11.15 TSizeF.equal(TSizeF,TSizeF):Boolean . . . 2009
77.11.16 TSizeF.notequal(TSizeF,TSizeF):Boolean . . . 2009
77.11.17 TSizeF.add(TSizeF,TSizeF):TSizeF . . . 2009
77.11.18 TSizeF.subtract(TSizeF,TSizeF):TSizeF . . . 2009
77.11.19 TSizeF.negative(TSizeF):TSizeF . . . 2009
77.11.20 TSizeF.multiply(TSizeF,single):TSizeF . . . 2009

77.11.21 TSizeF.multiply(single,TSizeF):TSizeF . . . 2009
77.11.22 TSizeF.assign(TPointF):TSizeF . . . 2010
77.11.23 TSizeF.assign(TSize):TSizeF . . . 2010
77.11.24 TSizeF.assign(TSizeF):TPointF . . . 2010
77.11.25 TSizeF.Width . . . 2010
77.11.26 TSizeF.Height . . . 2010
77.12 IClassFactory . . . 2010
77.12.1 Description . . . 2010
77.12.2 Method overview . . . 2010
77.12.3 IClassFactory.CreateInstance . . . 2010
77.12.4 IClassFactory.LockServer . . . 2011
77.13 ISequentialStream . . . 2011
77.13.1 Description . . . 2011
77.13.2 Method overview . . . 2011
77.13.3 ISequentialStream.Read . . . 2011
77.13.4 ISequentialStream.Write . . . 2011
77.14 IStream . . . 2012
77.14.1 Description . . . 2012
77.14.2 Method overview . . . 2012
77.14.3 IStream.Seek . . . 2012
77.14.4 IStream.SetSize . . . 2012
77.14.5 IStream.CopyTo . . . 2012
77.14.6 IStream.Commit . . . 2013
77.14.7 IStream.Revert . . . 2013
77.14.8 IStream.LockRegion . . . 2013
77.14.9 IStream.UnlockRegion . . . 2013
77.14.10 IStream.Stat . . . 2014
77.14.11 IStream.Clone . . . 2014
77.15 TBitConverter . . . 2014
77.15.1 Method overview . . . 2014
77.15.2 TBitConverter.UnsafeFrom . . . 2014
77.15.3 TBitConverter.From . . . 2014
77.15.4 TBitConverter.UnsafeInTo . . . 2015
77.15.5 TBitConverter.InTo . . . 2015

78 Reference for unit 'TypInfo' 2016
78.1 Used units . . . 2016
78.2 Overview . . . 2016
78.3 Auxiliary functions. . . . 2016
78.4 Getting or setting property values. . . . 2017

78.5 Examining published property information. . . . 2017
78.6 Constants, types and variables . . . 2018
78.6.1 Constants . . . 2018
78.6.2 Types . . . 2021
78.7 Procedures and functions . . . 2029
78.7.1 AddEnumElementAliases . . . 2029
78.7.2 AlignPTypeInfo . . . 2029
78.7.3 AlignTParamFlags . . . 2030
78.7.4 AlignTypeData . . . 2030
78.7.5 DerefTypeInfoPtr . . . 2030
78.7.6 FindPropInfo . . . 2030
78.7.7 GetDynArrayProp . . . 2031
78.7.8 GetEnumeratedAliasValue . . . 2032
78.7.9 GetEnumName . . . 2032
78.7.10 GetEnumNameCount . . . 2033
78.7.11 GetEnumProp . . . 2033
78.7.12 GetEnumValue . . . 2034
78.7.13 GetFloatProp . . . 2034
78.7.14 GetInt64Prop . . . 2035
78.7.15 GetInterfaceProp . . . 2036
78.7.16 GetMethodProp . . . 2036
78.7.17 GetObjectProp . . . 2038
78.7.18 GetObjectPropClass . . . 2039
78.7.19 GetOrdProp . . . 2040
78.7.20 GetPropInfo . . . 2041
78.7.21 GetPropInfos . . . 2041
78.7.22 GetPropList . . . 2042
78.7.23 GetPropValue . . . 2043
78.7.24 GetRawByteStrProp . . . 2044
78.7.25 GetRawInterfaceProp . . . 2044
78.7.26 GetSetProp . . . 2044
78.7.27 GetStrProp . . . 2046
78.7.28 GetTypeData . . . 2047
78.7.29 GetUnicodeStrProp . . . 2047
78.7.30 GetVariantProp . . . 2047
78.7.31 GetWideStrProp . . . 2048
78.7.32 IsPublishedProp . . . 2048
78.7.33 IsReadableProp . . . 2049
78.7.34 IsStoredProp . . . 2049
78.7.35 IsWriteableProp . . . 2050

78.7.36 PropIsType . . . 2050
78.7.37 PropType . . . 2051
78.7.38 RemoveEnumElementAliases . . . 2052
78.7.39 SetDynArrayProp . . . 2052
78.7.40 SetEnumProp . . . 2053
78.7.41 SetFloatProp . . . 2053
78.7.42 SetInt64Prop . . . 2053
78.7.43 SetInterfaceProp . . . 2054
78.7.44 SetMethodProp . . . 2054
78.7.45 SetObjectProp . . . 2055
78.7.46 SetOrdProp . . . 2055
78.7.47 SetPropValue . . . 2056
78.7.48 SetRawByteStrProp . . . 2056
78.7.49 SetRawInterfaceProp . . . 2056
78.7.50 SetSetProp . . . 2057
78.7.51 SetStrProp . . . 2057
78.7.52 SetToString . . . 2057
78.7.53 SetUnicodeStrProp . . . 2058
78.7.54 SetVariantProp . . . 2059
78.7.55 SetWideStrProp . . . 2059
78.7.56 StringToSet . . . 2059
78.8 TArrayTypeData . . . 2060
78.8.1 Property overview . . . 2060
78.8.2 TArrayTypeData.ElType . . . 2061
78.8.3 TArrayTypeData.Dims . . . 2061
78.9 TClassData . . . 2061
78.9.1 Property overview . . . 2061
78.9.2 TClassData.UnitName . . . 2061
78.9.3 TClassData.PropertyTable . . . 2062
78.10 TInterfaceData . . . 2062
78.10.1 Property overview . . . 2062
78.10.2 TInterfaceData.UnitName . . . 2063
78.10.3 TInterfaceData.PropertyTable . . . 2063
78.10.4 TInterfaceData.MethodTable . . . 2063
78.11 TInterfaceRawData . . . 2063
78.11.1 Property overview . . . 2064
78.11.2 TInterfaceRawData.UnitName . . . 2064
78.11.3 TInterfaceRawData.IIDStr . . . 2064
78.11.4 TInterfaceRawData.PropertyTable . . . 2064
78.11.5 TInterfaceRawData.MethodTable . . . 2065

78.12 TIntfMethodEntry . . . 2065
78.12.1 Property overview . . . 2066
78.12.2 TIntfMethodEntry.Name . . . 2066
78.12.3 TIntfMethodEntry.Param . . . 2066
78.12.4 TIntfMethodEntry.ResultLocs . . . 2066
78.12.5 TIntfMethodEntry.Tail . . . 2067
78.12.6 TIntfMethodEntry.Next . . . 2067
78.13 TIntfMethodTable . . . 2067
78.13.1 Property overview . . . 2067
78.13.2 TIntfMethodTable.Method . . . 2068
78.14 TManagedField . . . 2068
78.14.1 Property overview . . . 2068
78.14.2 TManagedField.TypeRef . . . 2068
78.15 TParameterLocation . . . 2068
78.15.1 Property overview . . . 2069
78.15.2 TParameterLocation.Reference . . . 2069
78.15.3 TParameterLocation.RegType . . . 2070
78.15.4 TParameterLocation.ShiftVal . . . 2070
78.16 TParameterLocations . . . 2070
78.16.1 Property overview . . . 2070
78.16.2 TParameterLocations.Location . . . 2071
78.16.3 TParameterLocations.Tail . . . 2071
78.17 TProcedureParam . . . 2071
78.17.1 Property overview . . . 2071
78.17.2 TProcedureParam.ParamType . . . 2072
78.17.3 TProcedureParam.Flags . . . 2072
78.18 TProcedureSignature . . . 2072
78.18.1 Method overview . . . 2072
78.18.2 Property overview . . . 2072
78.18.3 TProcedureSignature.GetParam . . . 2073
78.18.4 TProcedureSignature.ResultType . . . 2073
78.19 TPropData . . . 2073
78.19.1 Property overview . . . 2073
78.19.2 TPropData.Prop . . . 2074
78.19.3 TPropData.Tail . . . 2074
78.20 TPropInfo . . . 2074
78.20.1 Property overview . . . 2075
78.20.2 TPropInfo.PropType . . . 2075
78.20.3 TPropInfo.Tail . . . 2075
78.20.4 TPropInfo.Next . . . 2075

78.21 TTypeData . . . 2076
78.21.1 Property overview . . . 2091
78.21.2 TTypeData.BaseType . . . 2091
78.21.3 TTypeData.CompType . . . 2091
78.21.4 TTypeData.ParentInfo . . . 2091
78.21.5 TTypeData.RecInitData . . . 2092
78.21.6 TTypeData.HelperParent . . . 2092
78.21.7 TTypeData.ExtendedInfo . . . 2092
78.21.8 TTypeData.IntfParent . . . 2092
78.21.9 TTypeData.RawIntfParent . . . 2093
78.21.10 TTypeData.IIDStr . . . 2093
78.21.11 TTypeData.ElType2 . . . 2093
78.21.12 TTypeData.ElType . . . 2093
78.21.13 TTypeData.InstanceType . . . 2093
78.21.14 TTypeData.RefType . . . 2093
78.22 TVmtFieldEntry . . . 2094
78.22.1 Property overview . . . 2094
78.22.2 TVmtFieldEntry.Tail . . . 2094
78.22.3 TVmtFieldEntry.Next . . . 2094
78.23 TVmtFieldTable . . . 2095
78.23.1 Property overview . . . 2095
78.23.2 TVmtFieldTable.Field . . . 2095
78.24 TVmtMethodParam . . . 2095
78.24.1 Property overview . . . 2096
78.24.2 TVmtMethodParam.Name . . . 2096
78.24.3 TVmtMethodParam.Tail . . . 2096
78.24.4 TVmtMethodParam.Next . . . 2096
78.25 TVmtMethodTable . . . 2097
78.25.1 Property overview . . . 2097
78.25.2 TVmtMethodTable.Entry . . . 2097
78.26 EPropertyConvertError . . . 2097
78.26.1 Description . . . 2097
78.27 EPropertyError . . . 2097
78.27.1 Description . . . 2097

79 Reference for unit 'unicodedata' 2098
79.1 Used units . . . 2098
79.2 Overview . . . 2098
79.3 Constants, types and variables . . . 2099
79.3.1 Resource strings . . . 2099

79.3.2 Constants . . . 2099
79.3.3 Types . . . 2103
79.4 Procedures and functions . . . 2105
79.4.1 AddAliasCollation . . . 2105
79.4.2 BytesToName . . . 2105
79.4.3 BytesToString . . . 2105
79.4.4 CanonicalOrder . . . 2105
79.4.5 CompareSortKey . . . 2106
79.4.6 ComputeSortKey . . . 2106
79.4.7 FilterString . . . 2107
79.4.8 FindCollation . . . 2107
79.4.9 FreeCollation . . . 2107
79.4.10 FromUCS4 . . . 2107
79.4.11 GetCollationCount . . . 2108
79.4.12 GetProps . . . 2108
79.4.13 GetPropUCA . . . 2108
79.4.14 IncrementalCompareString . . . 2108
79.4.15 LoadCollation . . . 2109
79.4.16 NormalizeNFD . . . 2110
79.4.17 PrepareCollation . . . 2110
79.4.18 RegisterCollation . . . 2110
79.4.19 ToUCS4 . . . 2111
79.4.20 UnicodeIsHighSurrogate . . . 2111
79.4.21 UnicodeIsLowSurrogate . . . 2111
79.4.22 UnicodeIsSurrogatePair . . . 2112
79.4.23 UnicodeToLower . . . 2112
79.4.24 UnicodeToUpper . . . 2112
79.4.25 UnregisterCollation . . . 2113
79.4.26 UnregisterCollations . . . 2113
79.5 TCollationTable . . . 2113
79.5.1 Method overview . . . 2114
79.5.2 Property overview . . . 2114
79.5.3 TCollationTable.NormalizeName . . . 2114
79.5.4 TCollationTable.Clear . . . 2114
79.5.5 TCollationTable.IndexOf . . . 2114
79.5.6 TCollationTable.Find . . . 2114
79.5.7 TCollationTable.Add . . . 2115
79.5.8 TCollationTable.Remove . . . 2115
79.5.9 TCollationTable.Item . . . 2115
79.5.10 TCollationTable.Count . . . 2115

79.5.11 TCollationTable.Capacity . . . 2115
79.6 TUCA_DataBook . . . 2115
79.6.1 Method overview . . . 2116
79.6.2 TUCA_DataBook.IsVariable . . . 2116
79.7 TUCA_PropItemContextRec . . . 2116
79.7.1 Method overview . . . 2117
79.7.2 TUCA_PropItemContextRec.GetCodePoints . . . 2117
79.7.3 TUCA_PropItemContextRec.GetWeights . . . 2117
79.8 TUCA_PropItemContextTreeNodeRec . . . 2117
79.8.1 Method overview . . . 2117
79.8.2 TUCA_PropItemContextTreeNodeRec.GetLeftNode . . . 2117
79.8.3 TUCA_PropItemContextTreeNodeRec.GetRightNode . . . 2118
79.9 TUCA_PropItemContextTreeRec . . . 2118
79.9.1 Method overview . . . 2118
79.9.2 Property overview . . . 2118
79.9.3 TUCA_PropItemContextTreeRec.GetData . . . 2118
79.9.4 TUCA_PropItemContextTreeRec.Find . . . 2118
79.9.5 TUCA_PropItemContextTreeRec.Data . . . 2119
79.10 TUCA_PropItemRec . . . 2119
79.10.1 Method overview . . . 2120
79.10.2 Property overview . . . 2120
79.10.3 TUCA_PropItemRec.HasCodePoint . . . 2120
79.10.4 TUCA_PropItemRec.IsValid . . . 2120
79.10.5 TUCA_PropItemRec.GetWeightArray . . . 2120
79.10.6 TUCA_PropItemRec.GetSelfOnlySize . . . 2120
79.10.7 TUCA_PropItemRec.GetContextual . . . 2121
79.10.8 TUCA_PropItemRec.GetContext . . . 2121
79.10.9 TUCA_PropItemRec.IsDeleted . . . 2121
79.10.10 TUCA_PropItemRec.IsWeightCompress_1 . . . 2121
79.10.11 TUCA_PropItemRec.IsWeightCompress_2 . . . 2121
79.10.12 TUCA_PropItemRec.CodePoint . . . 2121
79.10.13 TUCA_PropItemRec.Contextual . . . 2122
79.11 TUC_Prop . . . 2122
79.11.1 Property overview . . . 2122
79.11.2 TUC_Prop.Category . . . 2123
79.11.3 TUC_Prop.WhiteSpace . . . 2123
79.11.4 TUC_Prop.HangulSyllable . . . 2123
79.11.5 TUC_Prop.NumericValue . . . 2123
79.12 TUInt24Rec . . . 2123
79.12.1 Method overview . . . 2127

79.12.2 TUInt24Rec.implicit(TUInt24Rec):Cardinal . . . 2128
79.12.3 TUInt24Rec.implicit(TUInt24Rec):LongInt . . . 2128
79.12.4 TUInt24Rec.implicit(TUInt24Rec):Word . . . 2128
79.12.5 TUInt24Rec.implicit(TUInt24Rec):Byte . . . 2128
79.12.6 TUInt24Rec.implicit(Cardinal):TUInt24Rec . . . 2129
79.12.7 TUInt24Rec.equal(TUInt24Rec,TUInt24Rec):Boolean . . . 2129
79.12.8 TUInt24Rec.equal(TUInt24Rec,Cardinal):Boolean . . . 2129
79.12.9 TUInt24Rec.equal(Cardinal,TUInt24Rec):Boolean . . . 2129
79.12.10 TUInt24Rec.equal(TUInt24Rec,LongInt):Boolean . . . 2129
79.12.11 TUInt24Rec.equal(LongInt,TUInt24Rec):Boolean . . . 2130
79.12.12 TUInt24Rec.equal(TUInt24Rec,Word):Boolean . . . 2130
79.12.13 TUInt24Rec.equal(Word,TUInt24Rec):Boolean . . . 2130
79.12.14 TUInt24Rec.equal(TUInt24Rec,Byte):Boolean . . . 2130
79.12.15 TUInt24Rec.equal(Byte,TUInt24Rec):Boolean . . . 2130
79.12.16 TUInt24Rec.notequal(TUInt24Rec,TUInt24Rec):Boolean . . . 2130
79.12.17 TUInt24Rec.notequal(TUInt24Rec,Cardinal):Boolean . . . 2131
79.12.18 TUInt24Rec.notequal(Cardinal,TUInt24Rec):Boolean . . . 2131
79.12.19 TUInt24Rec.greaterthan(TUInt24Rec,TUInt24Rec):Boolean . . . 2131
79.12.20 TUInt24Rec.greaterthan(TUInt24Rec,Cardinal):Boolean . . . 2131
79.12.21 TUInt24Rec.greaterthan(Cardinal,TUInt24Rec):Boolean . . . 2131
79.12.22 TUInt24Rec.greaterthanorequal(TUInt24Rec,TUInt24Rec):Boolean . . . 2132
79.12.23 TUInt24Rec.greaterthanorequal(TUInt24Rec,Cardinal):Boolean . . . 2132
79.12.24 TUInt24Rec.greaterthanorequal(Cardinal,TUInt24Rec):Boolean . . . 2132
79.12.25 TUInt24Rec.lessthan(TUInt24Rec,TUInt24Rec):Boolean . . . 2132
79.12.26 TUInt24Rec.lessthan(TUInt24Rec,Cardinal):Boolean . . . 2132
79.12.27 TUInt24Rec.lessthan(Cardinal,TUInt24Rec):Boolean . . . 2133
79.12.28 TUInt24Rec.lessthanorequal(TUInt24Rec,TUInt24Rec):Boolean . . . 2133
79.12.29 TUInt24Rec.lessthanorequal(TUInt24Rec,Cardinal):Boolean . . . 2133
79.12.30 TUInt24Rec.lessthanorequal(Cardinal,TUInt24Rec):Boolean . . . 2133

80 Reference for unit 'unicodeducet' 2134
80.1 Used units . . . 2134
80.2 Overview . . . 2134

81 Reference for unit 'Unix' 2135
81.1 Used units . . . 2135
81.2 Constants, types and variables . . . 2135
81.2.1 Constants . . . 2135
81.2.2 Types . . . 2141
81.3 Procedures and functions . . . 2150
81.3.1 AssignPipe . . . 2150

81.3.2 AssignStream . . . 2151
81.3.3 EpochToLocal . . . 2153
81.3.4 EpochToUniversal . . . 2153
81.3.5 FpExecL . . . 2153
81.3.6 FpExecLE . . . 2154
81.3.7 FpExecLP . . . 2155
81.3.8 FpExecLPE . . . 2156
81.3.9 FpExecV . . . 2156
81.3.10 FpExecVP . . . 2157
81.3.11 FpExecVPE . . . 2158
81.3.12 fpFlock . . . 2159
81.3.13 fpfStatFS . . . 2159
81.3.14 fpfsync . . . 2159
81.3.15 fpgettimeofday . . . 2160
81.3.16 fpStatFS . . . 2160
81.3.17 fpSystem . . . 2160
81.3.18 FSearch . . . 2161
81.3.19 GetDomainName . . . 2162
81.3.20 GetHostName . . . 2162
81.3.21 GetLocalTimezone . . . 2163
81.3.22 GetTimezoneFile . . . 2163
81.3.23 Gettzdaylight . . . 2163
81.3.24 GetTZInfo . . . 2163
81.3.25 GetTZInfoEx . . . 2164
81.3.26 Gettzname . . . 2164
81.3.27 GetTzseconds . . . 2164
81.3.28 GregorianToJulian . . . 2164
81.3.29 JulianToGregorian . . . 2164
81.3.30 LocalToEpoch . . . 2164
81.3.31 PClose . . . 2164
81.3.32 POpen . . . 2165
81.3.33 ReadTimezoneFile . . . 2166
81.3.34 RefreshTZInfo . . . 2166
81.3.35 ReReadLocalTime . . . 2166
81.3.36 SeekDir . . . 2166
81.3.37 SetTZInfo . . . 2167
81.3.38 TellDir . . . 2167
81.3.39 UniversalToEpoch . . . 2167
81.3.40 WaitProcess . . . 2167
81.3.41 WIFSTOPPED . . . 2168

81.3.42 W_EXITCODE . . . 2168
81.3.43 W_STOPCODE . . . 2168
81.4 TTZInfo . . . 2168
81.5 TTZInfoEx . . . 2169

82 Reference for unit 'unixcp' 2170
82.1 Used units . . . 2170
82.2 Overview . . . 2170
82.3 Constants, types and variables . . . 2170
82.3.1 Constants . . . 2170
82.3.2 Types . . . 2173
82.4 Procedures and functions . . . 2173
82.4.1 GetCodepageByName . . . 2173
82.4.2 GetCodepageData . . . 2174
82.4.3 GetSystemCodepage . . . 2174
82.5 TUnixCpData . . . 2174

83 Reference for unit 'unixtype' 2175
83.1 Used units . . . 2175
83.2 Overview . . . 2175
83.3 Constants, types and variables . . . 2175
83.3.1 Constants . . . 2175
83.3.2 Types . . . 2177
83.4 mbstate_t . . . 2187
83.5 pthread_attr_t . . . 2187
83.6 pthread_condattr_t . . . 2187
83.7 pthread_cond_t . . . 2188
83.8 pthread_mutexattr_t . . . 2188
83.9 pthread_rwlockattr_t . . . 2188
83.10 sched_param . . . 2188
83.11 sem_t . . . 2188
83.12 timespec . . . 2189
83.13 timeval . . . 2189
83.14 TStatfs . . . 2189
83.15 _pthread_fastlock . . . 2190

84 Reference for unit 'unixutil' 2191
84.1 Used units . . . 2191
84.2 Overview . . . 2191
84.3 Procedures and functions . . . 2191
84.3.1 ArrayStringToPPchar . . . 2191

84.3.2 StringToPPChar 2192

85 Reference for unit 'Variants' 2193

85.1 Used units 2193
85.2 Overview 2193
85.3 Constants, types and variables 2193
85.3.1 Constants 2193
85.3.2 Types 2194
85.3.3 Variables 2196
85.4 Procedures and functions 2198
85.4.1 DynArrayFromVariant 2198
85.4.2 DynArrayToVariant 2198
85.4.3 FindCustomVariantType 2198
85.4.4 FindVarData 2199
85.4.5 GetPropValue 2199
85.4.6 GetVariantProp 2199
85.4.7 HandleConversionException 2200
85.4.8 Null 2200
85.4.9 SetClearVarToEmptyParam 2200
85.4.10 SetPropValue 2200
85.4.11 SetVariantProp 2201
85.4.12 Unassigned 2201
85.4.13 VarArrayAsPSafeArray 2201
85.4.14 VarArrayCreate 2201
85.4.15 VarArrayCreateError 2202
85.4.16 VarArrayDimCount 2202
85.4.17 VarArrayHighBound 2203
85.4.18 VarArrayLock 2203
85.4.19 VarArrayLockedError 2203
85.4.20 VarArrayLowBound 2204
85.4.21 VarArrayOf 2204
85.4.22 VarArrayRef 2204
85.4.23 VarArrayUnlock 2204
85.4.24 VarAsError 2205
85.4.25 VarAsType 2205
85.4.26 VarBadIndexError 2205
85.4.27 VarBadTypeError 2205
85.4.28 VarCastError 2206
85.4.29 VarCastErrorOle 2206
85.4.30 VarCheckEmpty 2206

85.4.31 VarClear . . . 2206
85.4.32 VarCompareValue . . . 2207
85.4.33 VarCopyNoInd . . . 2207
85.4.34 VarEnsureRange . . . 2207
85.4.35 VarFromDateTime . . . 2207
85.4.36 VarInRange . . . 2208
85.4.37 VarInvalidArgError . . . 2208
85.4.38 VarInvalidNullOp . . . 2208
85.4.39 VarInvalidOp . . . 2208
85.4.40 VarIsArray . . . 2209
85.4.41 VarIsBool . . . 2209
85.4.42 VarIsByRef . . . 2209
85.4.43 VarIsClear . . . 2209
85.4.44 VarIsCustom . . . 2210
85.4.45 VarIsEmpty . . . 2210
85.4.46 VarIsEmptyParam . . . 2210
85.4.47 VarIsError . . . 2210
85.4.48 VarIsFloat . . . 2211
85.4.49 VarIsNull . . . 2211
85.4.50 VarIsNumeric . . . 2211
85.4.51 VarIsOrdinal . . . 2211
85.4.52 VarIsStr . . . 2212
85.4.53 VarIsType . . . 2212
85.4.54 VarNotImplError . . . 2212
85.4.55 VarOutOfMemoryError . . . 2212
85.4.56 VarOverflowError . . . 2213
85.4.57 VarRangeCheckError . . . 2213
85.4.58 VarResultCheck . . . 2213
85.4.59 VarSameValue . . . 2213
85.4.60 VarSupports . . . 2214
85.4.61 VarToDateTime . . . 2214
85.4.62 VarToStr . . . 2214
85.4.63 VarToStrDef . . . 2214
85.4.64 VarToUnicodeStr . . . 2215
85.4.65 VarToUnicodeStrDef . . . 2215
85.4.66 VarToWideStr . . . 2215
85.4.67 VarToWideStrDef . . . 2215
85.4.68 VarType . . . 2216
85.4.69 VarTypeAsText . . . 2216
85.4.70 VarTypeDeRef . . . 2216

85.4.71 VarTypeIsValidArrayType . . . 2216
85.4.72 VarTypeIsValidElementType . . . 2217
85.4.73 VarUnexpectedError . . . 2217
85.5 EVariantArrayCreateError . . . 2217
85.5.1 Description . . . 2217
85.6 EVariantArrayLockedError . . . 2217
85.6.1 Description . . . 2217
85.7 EVariantBadIndexError . . . 2218
85.7.1 Description . . . 2218
85.8 EVariantBadVarTypeError . . . 2218
85.8.1 Description . . . 2218
85.9 EVariantDispatchError . . . 2218
85.9.1 Description . . . 2218
85.10 EVariantInvalidArgError . . . 2218
85.10.1 Description . . . 2218
85.11 EVariantInvalidNullOpError . . . 2218
85.11.1 Description . . . 2218
85.12 EVariantInvalidOpError . . . 2218
85.12.1 Description . . . 2218
85.13 EVariantNotAnArrayError . . . 2219
85.13.1 Description . . . 2219
85.14 EVariantNotImplError . . . 2219
85.14.1 Description . . . 2219
85.15 EVariantOutOfMemoryError . . . 2219
85.15.1 Description . . . 2219
85.16 EVariantOverflowError . . . 2219
85.16.1 Description . . . 2219
85.17 EVariantParamNotFoundError . . . 2219
85.17.1 Description . . . 2219
85.18 EVariantRangeCheckError . . . 2219
85.18.1 Description . . . 2219
85.19 EVariantTypeCastError . . . 2220
85.19.1 Description . . . 2220
85.20 EVariantUnexpectedError . . . 2220
85.20.1 Description . . . 2220
85.21 IVarInstanceReference . . . 2220
85.21.1 Description . . . 2220
85.21.2 Method overview . . . 2220
85.21.3 IVarInstanceReference.GetInstance . . . 2220
85.22 IVarInvokeable . . . 2221

85.22.1 Description . . . 2221
85.22.2 Method overview . . . 2221
85.22.3 IVarInvokeable.DoFunction . . . 2221
85.22.4 IVarInvokeable.DoProcedure . . . 2221
85.22.5 IVarInvokeable.GetProperty . . . 2222
85.22.6 IVarInvokeable.SetProperty . . . 2222
85.23 TCustomVariantType . . . 2222
85.23.1 Description . . . 2222
85.23.2 Interfaces overview . . . 2223
85.23.3 Method overview . . . 2223
85.23.4 Property overview . . . 2223
85.23.5 TCustomVariantType.Create . . . 2223
85.23.6 TCustomVariantType.Destroy . . . 2223
85.23.7 TCustomVariantType.IsClear . . . 2224
85.23.8 TCustomVariantType.Cast . . . 2224
85.23.9 TCustomVariantType.CastTo . . . 2224
85.23.10 TCustomVariantType.CastToOle . . . 2224
85.23.11 TCustomVariantType.Clear . . . 2225
85.23.12 TCustomVariantType.Copy . . . 2225
85.23.13 TCustomVariantType.BinaryOp . . . 2225
85.23.14 TCustomVariantType.UnaryOp . . . 2226
85.23.15 TCustomVariantType.CompareOp . . . 2226
85.23.16 TCustomVariantType.Compare . . . 2226
85.23.17 TCustomVariantType.VarType . . . 2227
85.24 TInvokeableVariantType . . . 2227
85.24.1 Description . . . 2227
85.24.2 Interfaces overview . . . 2227
85.24.3 Method overview . . . 2227
85.24.4 TInvokeableVariantType.DoFunction . . . 2227
85.24.5 TInvokeableVariantType.DoProcedure . . . 2228
85.24.6 TInvokeableVariantType.GetProperty . . . 2228
85.24.7 TInvokeableVariantType.SetProperty . . . 2228
85.25 TPublishableVariantType . . . 2229
85.25.1 Description . . . 2229
85.25.2 Interfaces overview . . . 2229
85.25.3 Method overview . . . 2229
85.25.4 TPublishableVariantType.GetProperty . . . 2229
85.25.5 TPublishableVariantType.SetProperty . . . 2229

86 Reference for unit 'video' 2230

86.1 Used units . . . 2230
86.2 Overview . . . 2230
86.3 Examples utility unit. . . . 2231
86.4 Writing a custom video driver. . . . 2232
86.5 Constants, types and variables . . . 2235
86.5.1 Constants . . . 2235
86.5.2 Types . . . 2239
86.5.3 Variables . . . 2240
86.6 Procedures and functions . . . 2241
86.6.1 ClearScreen . . . 2241
86.6.2 DefaultErrorHandler . . . 2242
86.6.3 DoneVideo . . . 2242
86.6.4 GetCapabilities . . . 2242
86.6.5 GetCursorType . . . 2243
86.6.6 GetLockScreenCount . . . 2244
86.6.7 GetVideoDriver . . . 2245
86.6.8 GetVideoMode . . . 2245
86.6.9 GetVideoModeCount . . . 2246
86.6.10 GetVideoModeData . . . 2247
86.6.11 InitVideo . . . 2247
86.6.12 LockScreenUpdate . . . 2248
86.6.13 SetCursorPos . . . 2248
86.6.14 SetCursorType . . . 2249
86.6.15 SetVideoDriver . . . 2250
86.6.16 SetVideoMode . . . 2250
86.6.17 UnlockScreenUpdate . . . 2250
86.6.18 UpdateScreen . . . 2251
86.7 TVideoDriver . . . 2251
86.8 TVideoMode . . . 2252

87 Reference for unit 'WinCRT' 2253
87.1 Used units . . . 2253
87.2 Overview . . . 2253
87.3 Constants, types and variables . . . 2253
87.3.1 Variables . . . 2253
87.4 Procedures and functions . . . 2253
87.4.1 delay . . . 2253
87.4.2 keypressed . . . 2254
87.4.3 nosound . . . 2254
87.4.4 readkey . . . 2254

87.4.5 sound . . . 2254
87.4.6 textmode . . . 2255

88 Reference for unit 'WinDirs' 2256
88.1 Used units . . . 2256
88.2 Overview . . . 2256
88.3 Constants, types and variables . . . 2256
88.3.1 Constants . . . 2256
88.4 Procedures and functions . . . 2270
88.4.1 ConvertCSIDLtoFOLDERID . . . 2270
88.4.2 ConvertFOLDERIDtoCSIDL . . . 2270
88.4.3 GetWindowsSpecialDir . . . 2270
88.4.4 GetWindowsSpecialDirUnicode . . . 2270
88.4.5 GetWindowsSystemDirectory . . . 2270
88.4.6 GetWindowsSystemDirectoryUnicode . . . 2271

89 Reference for unit 'x86' 2272
89.1 Used units . . . 2272
89.2 Overview . . . 2272
89.3 Procedures and functions . . . 2272
89.3.1 fpIOperm . . . 2272
89.3.2 fpIoPL . . . 2273
89.3.3 ReadPort . . . 2273
89.3.4 ReadPortB . . . 2273
89.3.5 ReadPortL . . . 2274
89.3.6 ReadPortW . . . 2274
89.3.7 WritePort . . . 2274
89.3.8 WritePortB . . . 2275
89.3.9 WritePortl . . . 2275
89.3.10 WritePortW . . . 2275

About this guide

This document describes all constants, types, variables, functions and procedures as they are declared in the units that come standard with the Free Pascal Run-Time library (RTL).

Throughout this document, we will refer to functions, types and variables with `typewriter` font. Functions and procedures gave their own subsections, and for each function or procedure we have the following topics:

Declaration The exact declaration of the function.

Description What does the procedure exactly do ?

Errors What errors can occur.

See Also Cross references to other related functions/commands.

0.1 Overview

The Run-Time Library is the basis of all Free Pascal programs. It contains the basic units that most programs will use, and are made available on all platforms supported by Free pascal (well, more or less).

There are units for compatibility with the Turbo Pascal Run-Time library, and there are units for compatibility with Delphi.

On top of these two sets, there are also a series of units to handle keyboard/mouse and text screens in a cross-platform way.

Other units include platform specific units that implement the specifics of a platform, these are usually needed to support the Turbo Pascal or Delphi units.

Units that fall outside the above outline do not belong in the RTL, but should be included in the packages, or in the FCL.

Chapter 1

Reference for unit 'BaseUnix'

1.1 Used units

Table 1.1: Used units by unit 'BaseUnix'

Name	Page
System	1362
unixtype	2175

1.2 Overview

The BaseUnix unit was implemented by Marco Van de Voort. It contains basic Unix functionality. It supersedes the Linux unit of version 1.0.X of the compiler, but only implements a cleaned up, portable subset of that unit.

For porting FPC to new Unix-like platforms, it should be sufficient to implement the functionality in this unit for the new platform.

1.3 Constants, types and variables

1.3.1 Constants

```
ARG_MAX = UnixType.ARG_MAX
```

Maximum number of arguments to a program.

```
AT_EMPTY_PATH = $1000
```

Create link to old directy if path is empty.

```
AT_FDCWD = - 100
```

Flag for various *at calls to indicate current working directory.

```
AT_NO_AUTOMOUNT = $800
```

Do not auto mount filesystem if the base dir is a mount point.

```
AT_RECURSIVE = $8000
```

Recurse subdirectories when applying changes.

```
AT_REMOVEDIR = $200
```

Unlink at: specify rmdir behaviour.

```
AT_STATX_DONT_SYNC = $4000
```

Do not synchronize the attributes with the server (if any).

```
AT_STATX_FORCE_SYNC = $2000
```

Force the attributes to be synchronized with the server (if any).

```
AT_STATX_SYNC_AS_STAT = $0000
```

Behave as `fpStat`.

```
AT_STATX_SYNC_TYPE = $6000
```

Type of synchronisation required from `statx`.

```
AT_SYMLINK_FOLLOW = $400
```

Dereference symbolic links.

```
AT_SYMLINK_NOFOLLOW = $100
```

Do not dereference symbolic links.

```
BITSINWORD = 8 * sizeof(cuLong)
```

Number of bits in a word.

```
clone_flags_fork = $01200011
```

Request fork behaviour for clone call.

```
ESysE2BIG = 7
```

System error: Argument list too long.

```
ESysEACCES = 13
```

System error: Permission denied.

```
ESysEADDRINUSE = 98
```

System error: Address already in use.

```
ESysEADDRNOTAVAIL = 99
```

System error: Cannot assign requested address.

```
ESysEADV = 68
```

System error: Advertise error.

```
ESysEAFNOSUPPORT = 97
```

System error: Address family not supported by protocol.

```
ESysEAGAIN = 11
```

System error: Try again.

```
ESysEALREADY = 114
```

System error: Operation already in progress.

```
ESysEBADE = 52
```

System error: Invalid exchange.

```
ESysEBADF = 9
```

System error: Bad file number.

```
ESysEBADFD = 77
```

System error: File descriptor in bad state.

```
ESysEBADMSG = 74
```

System error: Not a data message.

```
ESysEBADR = 53
```

System error: Invalid request descriptor.

```
ESysEBADRQC = 56
```

System error: Invalid request code.

```
ESysEBADSLT = 57
```

System error: Invalid slot.

```
ESysEBFONT = 59
```

System error: Bad font file format.

```
ESysEBUSY = 16
```

System error: Device or resource busy.

```
ESysECANCELED = 125
```

Operation canceled.

```
ESysECHILD = 10
```

System error: No child processes.

```
ESysECHRNG = 44
```

System error: Channel number out of range.

```
ESysECOMM = 70
```

System error: Communication error on send.

```
ESysECONNABORTED = 103
```

System error: Software caused connection abort.

```
ESysECONNREFUSED = 111
```

System error: Connection refused.

```
ESysECONNRESET = 104
```

System error: Connection reset by peer.

```
ESysEDEADLK = 35
```

System error: Resource deadlock would occur.

```
ESysEDEADLOCK = ESysEDEADLK
```

System error: File locking deadlock error.

```
ESysEDESTADDRREQ = 89
```

System error: Destination address required.

```
ESysEDOM = 33
```

System error: Math argument out of domain of func.

```
ESysEDOTDOT = 73
```

System error: RFS specific error.

```
ESysEDQUOT = 122
```

System error: Quota exceeded.

```
ESysEEXIST = 17
```

System error: File exists.

```
ESysEFAULT = 14
```

System error: Bad address.

```
ESysEFBIG = 27
```

System error: File too large.

```
ESysEHOSTDOWN = 112
```

System error: Host is down.

```
ESysEHOSTUNREACH = 113
```

System error: No route to host.

```
ESysEIDRM = 43
```

System error: Identifier removed.

```
ESysEILSEQ = 84
```

System error: Illegal byte sequence.

```
ESysEINPROGRESS = 115
```

System error: Operation now in progress.

```
ESysEINTR = 4
```

System error: Interrupted system call.

```
ESysEINVAL = 22
```

System error: Invalid argument.

```
ESysEIO = 5
```

System error: I/O error.

```
ESysEISCONN = 106
```

System error: Transport endpoint is already connected.

```
ESysEISDIR = 21
```

System error: Is a directory.

```
ESysEISNAM = 120
```

System error: Is a named type file.

```
ESysEKEYEXPIRED = 127
```

Key has expired (Linux kernel module).

```
ESysEKEYREJECTED = 129
```

Key was rejected by service (Linux kernel module).

```
ESysEKEYREVOKED = 128
```

Key has been revoked (Linux kernel module).

```
ESysEL2HLT = 51
```

System error: Level 2 halted.

```
ESysEL2NSYNC = 45
```

System error: Level 2 not synchronized.

```
ESysEL3HLT = 46
```

System error: Level 3 halted.

```
ESysEL3RST = 47
```

System error: Level 3 reset.

```
ESysELIBACC = 79
```

System error: Can not access a needed shared library.

```
ESysELIBBAD = 80
```

System error: Accessing a corrupted shared library.

```
ESysELIBEXEC = 83
```

System error: Cannot exec a shared library directly.

```
ESysELIBMAX = 82
```

System error: Attempting to link in too many shared libraries.

```
ESysELIBSCN = 81
```

System error: .lib section in a.out corrupted.

```
ESysELNRNG = 48
```

System error: Link number out of range.

```
ESysELOOP = 40
```

System error: Too many symbolic links encountered.

```
ESysEMEDIUMTYPE = 124
```

Wrong medium type.

```
ESysEMFILE = 24
```

System error: Too many open files.

```
ESysEMLINK = 31
```

System error: Too many links.

```
ESysEMSGSIZE = 90
```

System error: Message too long.

```
ESysEMULTIHOP = 72
```

System error: Multihop attempted.

```
ESysENAMETOOLONG = 36
```

System error: File name too long.

```
ESysENAVAIL = 119
```

System error: No XENIX semaphores available.

```
ESysENETDOWN = 100
```

System error: Network is down.

```
ESysENETRESET = 102
```

System error: Network dropped connection because of reset.

```
ESysENETUNREACH = 101
```

System error: Network is unreachable.

```
ESysENFILE = 23
```

System error: File table overflow.

```
ESysENOANO = 55
```

System error: No anode.

```
ESysENOBUFS = 105
```

System error: No buffer space available.

```
ESysENOCSI = 50
```

System error: No CSI structure available.

```
ESysENODATA = 61
```

System error: No data available.

```
ESysENODEV = 19
```

System error: No such device.

```
ESysENOENT = 2
```

System error: No such file or directory.

```
ESysENOEXEC = 8
```

System error: Exec format error.

```
ESysENOKEY = 126
```

Required key not available (Linux kernel module).

```
ESysENOLCK = 37
```

System error: No record locks available.

```
ESysENOLINK = 67
```

System error: Link has been severed.

```
ESysENOMEDIUM = 123
```

No medium present.

```
ESysENOMEM = 12
```

System error: Out of memory.

```
ESysENOMSG = 42
```

System error: No message of desired type.

```
ESysENONET = 64
```

System error: Machine is not on the network.

```
ESysENOPKG = 65
```

System error: Package not installed.

```
ESysENOPROTOOPT = 92
```

System error: Protocol not available.

```
ESysENOSPC = 28
```

System error: No space left on device.

```
ESysENOSR = 63
```

System error: Out of streams resources.

```
ESysENOSTR = 60
```

System error: Device not a stream.

```
ESysENOSYS = 38
```

System error: Function not implemented.

```
ESysENOTBLK = 15
```

System error: Block device required.

```
ESysENOTCONN = 107
```

System error: Transport endpoint is not connected.

```
ESysENOTDIR = 20
```

System error: Not a directory.

```
ESysENOTEMPTY = 39
```

System error: Directory not empty.

```
ESysENOTNAM = 118
```

System error: Not a XENIX named type file.

```
ESysENOTRECOVERABLE = 131
```

State not recoverable (mutexes).

```
ESysENOTSOCK = 88
```

System error: Socket operation on non-socket.

```
ESysENOTTY = 25
```

System error: Not a typewriter.

```
ESysENOTUNIQ = 76
```

System error: Name not unique on network.

```
ESysENXIO = 6
```

System error: No such device or address.

```
ESysEOPNOTSUPP = 95
```

System error: Operation not supported on transport endpoint.

```
ESysEOVERFLOW = 75
```

System error: Value too large for defined data type.

```
ESysEOWNERDEAD = 130
```

Owner died (mutexes).

```
ESysEPERM = 1
```

System error: Operation not permitted.

```
ESysEPFNOSUPPORT = 96
```

System error: Protocol family not supported.

```
ESysEPIPE = 32
```

System error: Broken pipe.

```
ESysEPROTO = 71
```

System error: Protocol error.

```
ESysEPROTONOSUPPORT = 93
```

System error: Protocol not supported.

```
ESysEPROTOTYPE = 91
```

System error: Protocol wrong type for socket.

```
ESysERANGE = 34
```

System error: Math result not representable.

```
ESysEREMCHG = 78
```

System error: Remote address changed.

```
ESysEREMOTE = 66
```

System error: Object is remote.

```
ESysEREMOTEIO = 121
```

System error: Remote I/O error.

```
ESysERESTART = 85
```

System error: Interrupted system call should be restarted.

```
ESysERFKILL = 132
```

Operation not possible due to RF-Kill (wireless).

```
ESysEROFS = 30
```

System error: Read-only file system.

```
ESysESHUTDOWN = 108
```

System error: Cannot send after transport endpoint shutdown.

```
ESysESOCKTNOSUPPORT = 94
```

System error: Socket type not supported.

```
ESysESPIPE = 29
```

System error: Illegal seek.

```
ESysESRCH = 3
```

System error: No such process.

```
ESysESRMNT = 69
```

System error: Srmount error.

```
ESysESTALE = 116
```

System error: Stale NFS file handle.

```
ESysESTRPIPE = 86
```

System error: Streams pipe error.

```
ESysETIME = 62
```

System error: Timer expired.

```
ESysETIMEDOUT = 110
```

System error: Connection timed out.

```
ESysETOOMANYREFS = 109
```

System error: Too many references: cannot splice.

```
ESysETXTBSY = 26
```

System error: Text (code segment) file busy.

```
ESysEUCLEAN = 117
```

System error: Structure needs cleaning.

```
ESysEUNATCH = 49
```

System error: Protocol driver not attached.

```
ESysEUSERS = 87
```

System error: Too many users.

```
ESysEWOULDBLOCK = ESysEAGAIN
```

System error: Operation would block.

```
ESysEXDEV = 18
```

System error: Cross-device link.

```
ESysEXFULL = 54
```

System error: Exchange full.

```
FD_MAXFDSET = 1024
```

Maximum elements in a TFDSet (181) array.

```
FPE_FLTDIV = 3
```

Value signalling floating point divide by zero in case of SIGFPE signal.

```
FPE_FLTINV = 7
```

Value signalling floating point invalid operation in case of SIGFPE signal.

```
FPE_FLTOVF = 4
```

Value signalling floating point overflow in case of SIGFPE signal.

```
FPE_FLTRES = 6
```

Value signalling floating point inexact result in case of SIGFPE signal.

```
FPE_FLTSUB = 8
```

Value signalling floating point subscript out of range in case of SIGFPE signal.

```
FPE_FLTUND = 5
```

Value signalling floating point underflow in case of SIGFPE signal.

```
FPE_INTDIV = 1
```

Value signalling integer divide in case of SIGFPE signal.

```
FPE_INTOVF = 2
```

Value signalling integer overflow in case of SIGFPE signal.

```
F_GetFd = 1
```

fpFCntl (193) command: Get close-on-exec flag.

```
F_GetFl = 3
```

fpFCntl (193) command: Get file descriptor flags.

```
F_GetLk = 5
```

fpFCntl (193) command: Get lock.

```
F_GetOwn = 9
```

fpFCntl (193) command: get owner of file descriptor events.

```
F_OK = 0
```

fpAccess (184) call test: file exists.

```
F_SetFd = 2
```

fpFCntl (193) command: Set close-on-exec flag.

```
F_SetFl = 4
```

fpFCntl (193) command: Set file descriptor flags.

```
F_SetLk = 6
```

fpFCntl (193) command: Set lock.

```
F_SetLkW = 7
```

fpFCntl (193) command: Test lock.

```
F_SetOwn = 8
```

fpFCntl (193) command: Set owner of file descriptor events.

```
ln2bitmask = 1 shl ln2bitsinword - 1
```

Last bit in word.

```
ln2bitsinword = 6
```

Power of 2 number of bits in word.

```
MAP_ANON = MAP_ANONYMOUS
```

Anonymous memory mapping (data private to application).

```
MAP_ANONYMOUS = $20
```

FpMMap (207) map type: Don't use a file.

```
MAP_DENYWRITE = $800
```

FpMMap (207) option: Ignored.

```
MAP_EXECUTABLE = $1000
```

FpMMap (207) option: Ignored.

```
MAP_FAILED = pointer(- 1)
```

Memory mapping failed error code.

```
MAP_FIXED = $10
```

FpMMap (207) map type: Interpret addr exactly.

```
MAP_GROWSDOWN = $100
```

FpMMap (207) option: Memory grows downward (like a stack).

```
MAP_LOCKED = $2000
```

FpMMap (207) option: lock the pages in memory.

```
MAP_NORESERVE = $4000
```

FpMMap (207) option: Do not reserve swap pages for this memory.

```
MAP_PRIVATE = $2
```

FpMMap (207) map type: Changes are private.

```
MAP_SHARED = $1
```

FpMMap (207) map type: Share changes.

```
MAP_TYPE = $f
```

FpMMap (207) map type: Bitmask for type of mapping.

```
NAME_MAX = UnixType.NAME_MAX
```

Maximum filename length.

```
O_APPEND = $400
```

fpOpen (211) file open mode: Append to file.

```
O_CREAT = $40
```

fpOpen (211) file open mode: Create if file does not yet exist.

```
O_DIRECT = $4000
```

fpOpen (211) file open mode: Minimize caching effects.

```
O_DIRECTORY = $10000
```

fpOpen (211) file open mode: File must be directory.

```
O_EXCL = $80
```

fpOpen (211) file open mode: Open exclusively.

```
O_LARGEFILE = $8000
```

fpOpen (211) file open mode: Open for 64-bit I/O.

```
O_NDELAY = O_NONBLOCK
```

fpOpen (211) file open mode: Alias for O_NonBlock (161).

```
O_NOCTTY = $100
```

fpOpen (211) file open mode: No TTY control.

```
O_NOFOLLOW = $20000
```

fpOpen (211) file open mode: Fail if file is symbolic link.

```
O_NONBLOCK = $800
```

fpOpen (211) file open mode: Open in non-blocking mode.

```
O_RDONLY = 0
```

fpOpen (211) file open mode: Read only.

```
O_RDWR = 2
```

fpOpen (211) file open mode: Read/Write.

```
O_SYNC = $1000
```

fpOpen (211) file open mode: Write to disc at once.

```
O_TRUNC = $200
```

fpOpen (211) file open mode: Truncate file to length 0.

```
O_WRONLY = 1
```

fpOpen (211) file open mode: Write only.

```
PATH_MAX = UnixType.PATH_MAX
```

Maximum pathname length.

```
POLLERR = $0008
```

Error condition on output file descriptor.

```
POLLHUP = $0010
```

Hang up.

```
POLLIN = $0001
```

Data is available for reading.

```
POLLNVAL = $0020
```

Invalid request, file descriptor not open.

```
POLLOUT = $0004
```

Writing data will not block the write call.

```
POLLPRI = $0002
```

Urgent data is available for reading.

```
POLLRDBAND = $0080
```

Priority data ready for reading.

```
POLLRDNORM = $0040
```

Same as `POLLIN`.

```
POLLWRBAND = $0200
```

Priority data my be written.

```
POLLWRNORM = $0100
```

Equivalent to POLLOUT.

```
PRIO_PGRP = UnixType.PRIO_PGRP
```

Easy access alias for unixtype.PRIO_PGRP (2176).

```
PRIO_PROCESS = UnixType.PRIO_PROCESS
```

Easy access alias for unixtype.PRIO_PROCESS (2176).

```
PRIO_USER = UnixType.PRIO_USER
```

Easy access alias for unixtype.PRIO_USER (2176).

```
PROT_EXEC = $4
```

FpMMap (207) memory access: page can be executed.

```
PROT_NONE = $0
```

FpMMap (207) memory access: page can not be accessed.

```
PROT_READ = $1
```

FpMMap (207) memory access: page can be read.

```
PROT_WRITE = $2
```

FpMMap (207) memory access: page can be written.

```
RLIMIT_AS = 9
```

RLimit request address space limit.

```
RLIMIT_CORE = 4
```

RLimit request max core file size.

```
RLIMIT_CPU = 0
```

RLimit request CPU time in ms.

```
RLIMIT_DATA = 2
```

RLimit request max data size.

```
RLIMIT_FSIZE = 1
```

Rlimit request maximum file size.

```
RLIMIT_LOCKS = 10
```

RLimit request maximum file locks held.

```
RLIMIT_MEMLOCK = 8
```

RLimit request max locked-in-memory address space.

```
RLIMIT_NOFILE = 7
```

RLimit request max number of open files.

```
RLIMIT_NPROC = 6
```

RLimit request max number of processes.

```
RLIMIT_RSS = 5
```

RLimit request max resident set size.

```
RLIMIT_STACK = 3
```

RLimit request max stack size.

```
R_OK = 4
```

fpAccess (184) call test: read allowed.

```
SA_INTERRUPT = $20000000
```

Sigaction options: ?

```
SA_NOCLDSTOP = 1
```

Sigaction options: Do not receive notification when child processes stop.

```
SA_NOCLDWAIT = 2
```

Sigaction options: ?

```
SA_NODEFER = $40000000
```

Sigaction options: Do not mask signal in its own signal handler.

```
SA_NOMASK = SA_NODEFER
```

Sigaction options: Do not prevent the signal from being received when it is handled.

```
SA_ONESHOT = SA_RESETHAND
```

Sigaction options: Restore the signal action to the default state.

```
SA_ONSTACK = $08000000
```

SA_ONSTACK is used in the fpsigaction (223) to indicate the signal handler must be called on an alternate signal stack provided by sigaltstack(2) If an alternate stack is not available, the default stack will be used.

```
SA_RESETHAND = $80000000
```

Sigaction options: Restore signal action to default state when signal handler exits.

```
SA_RESTART = $10000000
```

Sigaction options: Provide behaviour compatible with BSD signal semantics.

```
SA_RESTORER = $04000000
```

Signal restorer handler.

```
SA_SIGINFO = 4
```

Sigaction options: The signal handler takes 3 arguments, not one.

```
SEEK_CUR = 1
```

fpLSeek (204) option: Set position relative to current position.

```
SEEK_END = 2
```

fpLSeek (204) option: Set position relative to end of file.

```
SEEK_SET = 0
```

fpLSeek (204) option: Set absolute position.

```
SIGABRT = 6
```

Signal: ABRT (Abort).

```
SIGALRM = 14
```

Signal: ALRM (Alarm clock).

```
SIGBUS = 7
```

Signal: BUS (bus error).

```
SIGCHLD = 17
```

Signal: CHLD (child status changed).

```
SIGCONT = 18
```

Signal: CONT (Continue).

```
SIGFPE = 8
```

Signal: FPE (Floating point error).

```
SIGHUP = 1
```

Signal: HUP (Hangup).

```
SIGILL = 4
```

Signal: ILL (Illegal instruction).

```
SIGINT = 2
```

Signal: INT (Interrupt).

```
SIGIO = 29
```

Signal: IO (I/O operation possible).

```
SIGIOT = 6
```

Signal: IOT (IOT trap).

```
SIGKILL = 9
```

Signal: KILL (unblockable).

```
SIGPIPE = 13
```

Signal: PIPE (Broken pipe.

```
SIGPOLL = SIGIO
```

Signal: POLL (Pollable event).

```
SIGPROF = 27
```

Signal: PROF (Profiling alarm).

```
SIGPWR = 30
```

Signal: PWR (power failure restart).

```
SIGQUIT = 3
```

Signal: QUIT.

```
SIGSEGV = 11
```

Signal: SEGV (Segmentation violation).

```
SIGSTKFLT = 16
```

Signal: STKFLT (Stack Fault).

```
SIGSTOP = 19
```

Signal: STOP (Stop, unblockable).

```
SIGTERM = 15
```

Signal: TERM (Terminate).

```
SIGTRAP = 5
```

Signal: TRAP (Trace trap).

```
SIGTSTP = 20
```

Signal: TSTP (keyboard stop).

```
SIGTTIN = 21
```

Signal: TTIN (Terminal input, background).

```
SIGTTOU = 22
```

Signal: TTOU (Terminal output, background).

```
SIGUNUSED = 31
```

Signal: Unused.

```
SIGURG = 23
```

Signal: URG (Socket urgent condition).

```
SIGUSR1 = 10
```

Signal: USR1 (User-defined signal 1).

```
SIGUSR2 = 12
```

Signal: USR2 (User-defined signal 2).

```
SIGVTALRM = 26
```

Signal: VTALRM (Virtual alarm clock).

```
SIGWINCH = 28
```

Signal: WINCH (Window/Terminal size change).

```
SIGXCPU = 24
```

Signal: XCPU (CPU limit exceeded).

```
SIGXFSZ = 25
```

Signal: XFSZ (File size limit exceeded).

```
SIG_BLOCK = 0
```

Sigprocmask flags: Add signals to the set of blocked signals.

```
SIG_DFL = 0
```

Signal handler: Default signal handler.

```
SIG_ERR = - 1
```

Signal handler: error.

```
SIG_IGN = 1
```

Signal handler: Ignore signal.

```
SIG_MAXSIG = UnixType.SIG_MAXSIG
```

Maximum system signal number.

```
SIG_SETMASK = 2
```

Sigprocmask flags: Set of blocked signals is given.

```
SIG_UNBLOCK = 1
```

Sigprocmask flags: Remove signals from the set set of blocked signals.

```
SI_PAD_SIZE = 128 div sizeof(longint) - 3
```

Signal information pad size.

```
SYS_NMLN = UnixType.SYS_NMLN
```

Max system name length.

```
S_IFBLK = 24576
```

File (#rtl.baseunix.stat (240) record) mode: Block device.

```
S_IFCHR = 8192
```

File (#rtl.baseunix.stat (240) record) mode: Character device.

```
S_IFDIR = 16384
```

File (#rtl.baseunix.stat (240) record) mode: Directory.

```
S_IFIFO = 4096
```

File (#rtl.baseunix.stat (240) record) mode: FIFO.

```
S_IFLNK = 40960
```

File (#rtl.baseunix.stat (240) record) mode: Link.

```
S_IFMT = 61440
```

File (#rtl.baseunix.stat (240) record) mode: File type bit mask.

```
S_IFREG = 32768
```

File (#rtl.baseunix.stat (240) record) mode: Regular file.

```
S_IFSOCK = 49152
```

File (#rtl.baseunix.stat (240) record) mode: Socket.

```
S_IRGRP = %0000100000
```

Mode flag: Read by group.

```
S_IROTH = %0000000100
```

Mode flag: Read by others.

```
S_IRUSR = %0100000000
```

Mode flag: Read by owner.

```
S_IRWXG = S_IRGRP or S_IWGRP or S_IXGRP
```

Mode flag: Read, write, execute by groups.

```
S_IRWXO = S_IROTH or S_IWOTH or S_IXOTH
```

Mode flag: Read, write, execute by others.

```
S_IRWXU = S_IRUSR or S_IWUSR or S_IXUSR
```

Mode flag: Read, write, execute by user.

```
S_ISGID = &2000
```

Mode flag: Set Group ID on execution.

```
S_ISUID = &4000
```

Mode flag: Set user ID on execution.

```
S_ISVTX = &1000
```

Mode flag: Set sticky bit.

```
S_IWGRP = %0000010000
```

Mode flag: Write by group.

```
S_IWOTH = %0000000010
```

Mode flag: Write by others.

```
S_IWUSR = %0010000000
```

Mode flag: Write by owner.

```
S_IXGRP = %0000001000
```

Mode flag: Execute by group.

```
S_IXOTH = %0000000001
```

Mode flag: Execute by others.

```
S_IXUSR = %0001000000
```

Mode flag: Execute by owner.

```
UTSNAME_DOMAIN_LENGTH = UTSNAME_LENGTH
```

Max length of utsname (246) domain name.

```
UTSNAME_LENGTH = SYS_NMLN
```

Max length of utsname (246) system name, release, version, machine.

```
UTSNAME_NODENAME_LENGTH = UTSNAME_LENGTH
```

Max length of utsname (246) node name.

```
WNOHANG = 1
```

#rtl.baseunix.fpWaitpid (236) option: Do not wait for processes to terminate.

```
wordsinfdset = FD_MAXFDSET div BITSINWORD
```

Number of words in a TFDSet (181) array.

```
wordsinsigset = SIG_MAXSIG div BITSINWORD
```

Number of words in a signal set.

```
WUNTRACED = 2
```

#rtl.baseunix.fpWaitpid (236) option: Also report children which were stopped but not yet reported.

```
W_OK = 2
```

fpAccess (184) call test: write allowed.

```
X_OK = 1
```

fpAccess (184) call test: execute allowed.

```
_STAT_VER = _STAT_VER_LINUX
```

Stat version number.

```
_STAT_VER_KERNEL = 0
```

Current version of stat record.

```
_STAT_VER_LINUX = 1
```

Version of Linux stat record.

1.3.2 Types

```
Blkcnt64_t = cuint64
```

64-bit block count.

```
Blkcnt_t = cuint
```

Block count type.

```
Blksize_t = cuint
```

Block size type.

```
cbool = UnixType.cbool
```

Boolean type.

```
cchar = UnixType.cchar
```

Alias for #rtl.UnixType.cchar (2177).

```
cdouble = UnixType.cdouble
```

Double precision real format.

```
cfloat = UnixType.cfloat
```

Floating-point real format.

```
cint = UnixType.cint
```

C type: integer (natural size).

```
cint16 = UnixType.cint16
```

C type: 16 bits sized, signed integer.

```
cint32 = UnixType.cint32
```

C type: 32 bits sized, signed integer.

```
cint64 = UnixType.cint64
```

C type: 64 bits sized, signed integer.

```
cint8 = UnixType.cint8
```

C type: 8 bits sized, signed integer.

```
clock_t = UnixType.clock_t
```

Clock ticks type.

```
clong = UnixType.clong
```

C type: long signed integer (double sized).

```
clonglong = UnixType.clonglong
```

C type: 64-bit (double long) signed integer.

```
coff_t = UnixType.TOff
```

Character offset type.

```
cschar = UnixType.cschar
```

Signed character type.

```
cshort = UnixType.cshort
```

C type: short signed integer (half sized).

```
csigned = UnixType.csigned
```

`csigned` is an alias for cint (171).

```
csint = UnixType.csint
```

Signed integer.

```
csize_t = UnixType.size_t
```

Character size type.

```
cslong = UnixType.cslong
```

The size is CPU dependent.

```
cslonglong = UnixType.cslonglong
```

`cslonglong` is an alias for clonglong (172).

```
csshort = UnixType.csshort
```

Short signed integer type.

```
cuchar = UnixType.cuchar
```

Alias for #rtl.UnixType.cuchar (2178).

```
cuint = UnixType.cuint
```

C type: unsigned integer (natural size).

```
cuint16 = UnixType.cuint16
```

C type: 16 bits sized, unsigned integer.

```
cuint32 = UnixType.cuint32
```

C type: 32 bits sized, unsigned integer.

```
cuint64 = UnixType.cuint64
```

C type: 64 bits sized, unsigned integer.

```
cuint8 = UnixType.cuint8
```

C type: 8 bits sized, unsigned integer.

```
culong = UnixType.culong
```

C type: long unsigned integer (double sized).

```
culonglong = UnixType.culonglong
```

C type: 64-bit (double long) unsigned integer.

```
cunsigned = UnixType.cunsigned
```

Alias for #rtl.unixtype.cunsigned (2179).

```
cushort = UnixType.cushort
```

C type: short unsigned integer (half sized).

```
dev_t = UnixType.dev_t
```

Device descriptor type.

```
gid_t = UnixType.gid_t
```

Group ID type.

```
ino_t = UnixType.ino_t
```

Inode type.

```
kernel_gid_t = cuint
```

`kernel_gid_t` may differ from the libc type used to describe group IDs.

```
kernel_loff_t = clonglong
```

Long kernel offset type.

```
kernel_mode_t = cuint
```

`kernel_mode_t` may differ from the libc type used to describe file modes.

```
kernel_off_t = clong
```

Kernel offset type.

```
kernel_uid_t = cuint
```

`kernel_uid_t` may differ from the libc type used to describe user IDs.

```
mode_t = UnixType.mode_t
```

Inode mode type.

```
nlink_t = UnixType.nlink_t
```

Number of links type.

```
off_t = UnixType.off_t
```

Offset type.

```
PBlkCnt = ^Blkcnt_t
```

pointer to TBlkCnt (181) type.

```
PBlkSize = ^Blksize_t
```

Pointer to TBlkSize (181) type.

```
pcbool = UnixType.pcbool
```

Pointer to boolean type cbool (171)

```
pcchar = UnixType.pcchar
```

Alias for #rtl.UnixType.pcchar (2180).

```
pcdouble = UnixType.pcdouble
```

Pointer to cdouble (171) type.

```
pcfloat = UnixType.pcfloat
```

Pointer to cfloat (171) type.

```
pcint = UnixType.pcint
```

Pointer to cInt (171) type.

```
pcint16 = UnixType.pcint16
```

Pointer to 16-bit signed integer type.

```
pcint32 = UnixType.pcint32
```

Pointer to signed 32-bit integer type.

```
pcint64 = UnixType.pcint64
```

Pointer to signed 64-bit integer type.

```
pcint8 = UnixType.pcint8
```

Pointer to 8-bits signed integer type.

```
pClock = UnixType.pClock
```

Pointer to TClock (181) type.

```
pclong = UnixType.pclong
```

Pointer to cLong (172) type.

```
pclonglong = UnixType.pclonglong
```

Pointer to longlong type.

```
pcschar = UnixType.pcschar
```

Pointer to character type cschar (172).

```
pcshort = UnixType.pcshort
```

Pointer to cShort (172) type.

```
pcsigned = UnixType.pcsigned
```

Pointer to signed integer type csigned (172).

```
pcsint = UnixType.pcsint
```

Pointer to signed integer type csint (172)

```
pcsize_t = UnixType.psize_t
```

Pointer to `csize_t`.

```
pcslong = UnixType.pcslong
```

Pointer to the signed long cslong (172)

```
pcslonglong = UnixType.pcslonglong
```

Pointer to Signed longlong type cslonglong (172)

```
pcsshort = UnixType.pcsshort
```

Pointer to short signed integer type csshort (172)

```
pcuchar = UnixType.pcuchar
```

Alias for #rtl.UnixType.pcuchar (2181).

```
pcuint = UnixType.pcuint
```

Pointer to cUInt (173) type.

```
pcuint16 = UnixType.pcuint16
```

Pointer to 16-bit unsigned integer type.

```
pcuint32 = UnixType.pcuint32
```

Pointer to unsigned 32-bit integer type.

```
pcuint64 = UnixType.pcuint64
```

Pointer to unsigned 64-bit integer type.

```
pcuint8 = UnixType.pcuint8
```

Pointer to 8-bits unsigned integer type.

```
pculong = UnixType.pculong
```

Pointer to cuLong (173) type.

```
pculonglong = UnixType.pculonglong
```

Unsigned longlong type.

```
pcunsigned = UnixType.pcunsigned
```

Alias for #rtl.unixtype.pcunsigned (2182).

```
pcushort = UnixType.pcushort
```

Pointer to cuShort (173) type.

```
pDev = UnixType.pDev
```

Pointer to TDev (181) type.

```
pDir = ^Dir
```

Pointer to TDir (181) record.

```
pDirent = ^Dirent
```

Pointer to TDirent (181) record.

```
pFDSet = ^TFDSet
```

Pointer to TFDSet (181) type.

```
pFilDes = ^TFilDes
```

Pointer to TFilDes (181) type.

```
pfpstate = ^tfpstate
```

Pointer to tfpstate (241) record.

```
pGid = UnixType.pGid
```

Pointer to TGid (181) type.

```
pGrpArr = ^TGrpArr
```

Pointer to TGrpArr (181) array.

```
pid_t = UnixType.pid_t
```

Process ID type.

```
pIno = UnixType.pIno
```

Pointer to TIno (182) type.

```
piovec = ^tiovec
```

pointer to a iovec (239) record.

```
pMode = UnixType.pMode
```

Pointer to TMode (182) type.

```
pnLink = UnixType.pnLink
```

Pointer to TnLink (182) type.

```
pOff = UnixType.pOff
```

Pointer to TOff (182) type.

```
pPid = UnixType.pPid
```

Pointer to TPid (182) type.

```
ppollfd = ^pollfd
```

Pointer to `tpollfd`.

```
PRLimit = ^TRLimit
```

Pointer to TRLimit (243) record.

```
psigactionrec = ^sigactionrec
```

Pointer to SigActionRec (240) record type.

```
PSigContext = ^TSigContext
```

Pointer to #rtl.baseunix.TSigContext (244) record type.

```
psiginfo = ^tsiginfo
```

Pointer to #rtl.baseunix.TSigInfo (245) record type.

```
psigset = ^tsigset
```

Pointer to SigSet (180) type.

```
pSize = UnixType.pSize
```

Pointer to TSize (183) type.

```
pSize_t = UnixType.pSize_t
```

Pointer to `Size_t`.

```
pSocklen = UnixType.pSocklen
```

Pointer to TSockLen (183) type.

```
psSize = UnixType.psSize
```

Pointer to TsSize (183) type.

```
PStat = ^Stat
```

Pointer to TStat (183) type.

```
pstatfs = UnixType.PStatFs
```

This is an alias for the type defined in the #rtl.unixtype (2175) unit.

```
pthread_cond_t = UnixType.pthread_cond_t
```

Thread conditional variable type.

```
pthread_mutex_t = UnixType.pthread_mutex_t
```

Thread mutex type.

```
pthread_t = UnixType.pthread_t
```

POSIX thread type.

```
pTime = UnixType.pTime
```

Pointer to TTime (183) type.

```
ptimespec = UnixType.ptimespec
```

Pointer to timespec (181) type.

```
ptimeval = UnixType.ptimeval
```

Pointer to timeval (181) type.

```
ptimezone = ^timezone
```

Pointer to TimeZone (242) record.

```
ptime_t = UnixType.ptime_t
```

Pointer to time_t (182) type.

```
PTms = ^tms
```

Pointer to TTms (183) type.

```
Pucontext = ^Tucontext
```

Pointer to TUContext (245) type.

```
pUid = UnixType.pUid
```

Pointer to TUid (183) type.

```
pUtimBuf = ^UTimBuf
```

Pointer to TUTimBuf (183) type.

```
PUtsName = ^TUtsName
```

Pointer to TUtsName (183) type.

```
rlim_t = culong
```

rlim_t is used as the type for the various fields in the TRLimit (243) record.

```
sigactionhandler = sigactionhandler_t
```

When installing a signal handler, the actual signal handler must be of type SigActionHandler.

```
sigactionhandler_t = procedure(signal: LongInt; info: psiginfo;
  context: PSigContext)
```

Standard signal action handler prototype.

```
signalhandler = signalhandler_t
```

Simple signal handler prototype.

```
signalhandler_t = procedure(signal: LongInt)
```

Standard signal handler prototype.

```
sigrestorerhandler = sigrestorerhandler_t
```

Alias for sigrestorerhandler_t (180) type.

```
sigrestorerhandler_t = procedure
```

Standard signal action restorer prototype.

```
sigset = sigset_t
```

Signal set type.

```
sigset_t = Array[0..wordsinsigset-1] of culong
```

Signal set type.

```
size_t = UnixType.size_t
```

Size specification type.

```
socklen_t = UnixType.socklen_t
```

Socket address length type.

```
ssize_t = UnixType.ssize_t
```

Small size type.

```
TBlkCnt = Blkcnt_t
```

Alias for Blkcnt_t (171) type.

```
TBlkSize = Blksize_t
```

Alias for blksize_t (171) type.

```
TClock = UnixType.TClock
```

Alias for clock_t (172) type.

```
TDev = UnixType.TDev
```

Alias for dev_t (173) type.

```
TDir = Dir
```

Alias for Dir (238) type.

```
TDirent = Dirent
```

Alias for Dirent (239) type.

```
TFDSet = Array[0..(FD_MAXFDSETdivBITSINWORD)-1] of TFDSetEl
```

File descriptor set for fpSelect (219) call.

```
TFDSetEl = culong
```

Type alias for an element of the TFDSet.

```
TFilDes = Array[0..1] of cint
```

Array of file descriptors as used in fpPipe (213) call.

```
TGid = UnixType.TGid
```

Alias for gid_t (173) type.

```
TGrpArr = Array[0..0] of TGid
```

Array of gid_t (173) IDs.

```
timespec = UnixType.timespec
```

Short time specification type.

```
timeval = UnixType.timeval
```

Time specification type.

```
time_t = UnixType.time_t
```

Time span type.

```
TIno = UnixType.TIno
```

Alias for ino_t (173) type.

```
TIOCtlRequest = UnixType.TIOCtlRequest
```

Easy access alias for unixtype.TIOCtlRequest (2185).

```
tiovec = iovec
```

Alias for the iovec (239) record type.

```
TMode = UnixType.TMode
```

Alias for mode_t (174) type.

```
TnLink = UnixType.TnLink
```

Alias for nlink_t (174) type.

```
TOff = UnixType.TOff
```

Alias for off_t (174) type.

```
TPid = UnixType.TPid
```

Alias for pid_t (177) type.

```
tpollfd = pollfd
```

Alias for `pollfd` type.

```
tsigactionhandler = sigactionhandler_t
```

Alias for sigactionhandler_t (180) type.

```
tsignalhandler = signalhandler_t
```

Alias for signalhandler_t (180) type.

```
tsigrestorerhandler = sigrestorerhandler_t
```

Alias for sigrestorerhandler_t (180) type.

```
tsigset = sigset_t
```

Alias for SigSet (180) type.

```
TSize = UnixType.TSize
```

Alias for size_t (180) type.

```
TSocklen = UnixType.TSocklen
```

Alias for socklen_t (180) type.

```
TsSize = UnixType.TsSize
```

Alias for ssize_t (180) type.

```
TStat = Stat
```

Alias for Stat (240) type.

```
tstatfs = UnixType.TStatFs
```

Record describing a file system in the unix.fpstatfs (146) call.

```
TTime = UnixType.TTime
```

Alias for TTime (183) type.

```
Ttimespec = UnixType.Ttimespec
```

Alias for TimeSpec (181) type.

```
TTimeVal = UnixType.TTimeVal
```

Alias for timeval (181) type.

```
TTimeZone = timezone
```

Alias for TimeZone (242) record.

```
TTms = tms
```

Alias for Tms (242) record type.

```
TUid = UnixType.TUid
```

Alias for uid_t (183) type.

```
TUtimBuf = UTimBuf
```

Alias for UtimBuf (245) type.

```
TUtsName = UtsName
```

Alias for UtsName (246) type.

```
uid_t = UnixType.uid_t
```

User ID type.

1.4 Procedures and functions

1.4.1 CreateShellArgV

Synopsis: Create a null-terminated array of strings from a command-line string.

Declaration:
```
function CreateShellArgV(const prog: string) : PPChar
function CreateShellArgV(const prog: RawByteString) : PPChar
```

Visibility: default

Description: `CreateShellArgV` creates a command-line string for executing a shell command using 'sh -c'. The result is a null-terminated array of null-terminated strings suitable for use in fpExecv (191) and friends.

Errors: If no more memory is available, a heap error may occur.

See also: fpExecv (191), FreeShellArgV (237)

1.4.2 FpAccess

Synopsis: Check file access.

Declaration:
```
function FpAccess(pathname: PChar; aMode: cint) : cint
function FpAccess(const pathname: RawByteString; aMode: cint) : cint
```

Visibility: default

Description: `FpAccess` tests user's access rights on the specified file. `Mode` is a mask existing of one or more of the following:

R_OK User has read rights.

W_OK User has write rights.

X_OK User has execute rights.

F_OK File exists.

The test is done with the real user ID, instead of the effective user ID. If the user has the requested rights, zero is returned. If access is denied, or an error occurred, a nonzero value is returned.

Errors: Extended error information can be retrieved using fpGetErrno (198).

sys_eaccess The requested access is denied, either to the file or one of the directories in its path.

sys_einval `Mode` was incorrect.

sys_enoent A directory component in `Path` doesn't exist or is a dangling symbolic link.

sys_enotdir A directory component in `Path` is not a directory.

sys_enomem Insufficient kernel memory.

sys_eloop `Path` has a circular symbolic link.

See also: FpChown (187), FpChmod (186)

Listing: ./examples/bunixex/ex26.pp

```
Program Example26;

{ Program to demonstrate the Access function. }

Uses BaseUnix;

begin
  if fpAccess ('/etc/passwd',W_OK)=0 then
    begin
    Writeln ('Better check your system.');
    Writeln ('I can write to the /etc/passwd file !');
    end;
end.
```

1.4.3 FpAlarm

Synopsis: Schedule an alarm signal to be delivered.

Declaration: `function FpAlarm(seconds: cuint) : cuint`

Visibility: default

Description: `FpAlarm` schedules an alarm signal to be delivered to your process in `Seconds` seconds. When `Seconds` seconds have elapsed, the system will send a `SIGALRM` signal to the current process. If `Seconds` is zero, then no new alarm will be set. Whatever the value of `Seconds`, any previous alarm is cancelled.

The function returns the number of seconds till the previously scheduled alarm was due to be delivered, or zero if there was none. A negative value indicates an error.

See also: fpSigAction (223), fpPause (213)

Listing: ./examples/bunixex/ex59.pp

```
Program Example59;

{ Program to demonstrate the Alarm function. }

Uses BaseUnix;

Procedure AlarmHandler(Sig : cint);cdecl;

begin
  Writeln ('Got to alarm handler');
end;

begin
  Writeln('Setting alarm handler');
  fpSignal(SIGALRM, SignalHandler(@AlarmHandler));
  Writeln ('Scheduling Alarm in 10 seconds');
  fpAlarm(10);
  Writeln ('Pausing');
  fpPause;
  Writeln ('Pause returned');
end.
```

1.4.4 FpChdir

Synopsis: Change current working directory.

Declaration:
```
function FpChdir(path: PChar) : cint
function FpChdir(const path: RawByteString) : cint
```

Visibility: default

Description: `fpChDir` sets the current working directory to `Path`.

It returns zero if the call was successful, -1 on error.

Note: There exist a portable alternative to fpChDir: system.chdir. Please use fpChDir only if you are writing Unix specific code. System.chdir will work on all operating systems.

Errors: Extended error information can be retrieved using fpGetErrno (198).

See also: fpGetCwd (197)

1.4.5 FpChmod

Synopsis: Change file permission bits.

Declaration:
```
function FpChmod(path: PChar; Mode: TMode) : cint
function FpChmod(const path: RawByteString; Mode: TMode) : cint
```

Visibility: default

Description: `fpChmod` sets the Mode bits of the file in `Path` to `Mode`. `Mode` can be specified by 'or'-ing the following values:

S_ISUID Set user ID on execution.

S_ISGID Set Group ID on execution.

S_ISVTX Set sticky bit.

S_IRUSR Read by owner.

S_IWUSR Write by owner.

S_IXUSR Execute by owner.

S_IRGRP Read by group.

S_IWGRP Write by group.

S_IXGRP Execute by group.

S_IROTH Read by others.

S_IWOTH Write by others.

S_IXOTH Execute by others.

S_IRWXO Read, write, execute by others.

S_IRWXG Read, write, execute by groups.

S_IRWXU Read, write, execute by user.

If the function is successful, zero is returned. A nonzero return value indicates an error.

Errors: The following error codes are returned:

sys_eperm The effective UID doesn't match the ownership of the file, and is not zero. Owner or group were not specified correctly.

sys_eaccess One of the directories in `Path` has no search (=execute) permission.

sys_enoentA directory entry in `Path` does not exist or is a symbolic link pointing to a non-existent directory.

sys_enomem Insufficient kernel memory.

sys_erofs The file is on a read-only file system.

sys_eloop`Path` has a reference to a circular symbolic link, i.e. a symbolic link, whose expansion points to itself.

See also: fpChown (187), fpAccess (184)

Listing: ./examples/bunixex/ex23.pp

```
Program Example23;

{ Program to demonstrate the Chmod function. }

Uses BaseUnix,Unix;

Var F : Text;

begin
  { Create a file }
  Assign (f,'testex21');
  Rewrite (F);
  Writeln (f,'#!/bin/sh');
  Writeln (f,'echo Some text for this file');
  Close (F);
  fpChmod ('testex21',&777);
  { File is now executable  }
  fpexecl ('./testex21',[]);
end.
```

1.4.6 FpChown

Synopsis: Change owner of file.

Declaration:
```
function FpChown(path: PChar; owner: TUid; group: TGid) : cint
function FpChown(const path: RawByteString; owner: TUid; group: TGid)
                 : cint
```

Visibility: default

Description: `fpChown` sets the User ID and Group ID of the file in `Path` to `Owner`,`Group`.

The function returns zero if the call was successful, a nonzero return value indicates an error.

Errors: The following error codes are returned:

sys_eperm The effective UID doesn't match the ownership of the file, and is not zero. Owner or group were not specified correctly.

sys_eaccess One of the directories in `Path` has no search (=execute) permission.

sys_enoentA directory entry in `Path` does not exist or is a symbolic link pointing to a non-existent directory.

sys_enomem Insufficient kernel memory.

sys_erofs The file is on a read-only file system.

sys_eloop`Path` has a reference to a circular symbolic link, i.e. a symbolic link, whose expansion points to itself.

See also: fpChmod (186), fpAccess (184)

Listing: ./examples/bunixex/ex24.pp

```
Program Example24;

{ Program to demonstrate the Chown function. }

Uses BaseUnix;

Var UID : TUid;
    GID : TGid;
    F : Text;

begin

  Writeln ('This will only work if you are root.');
  Write ('Enter a UID : ');readln(UID);
  Write ('Enter a GID : ');readln(GID);
  Assign (f,'test.txt');
  Rewrite (f);
  Writeln (f,'The owner of this file should become : ');
  Writeln (f,'UID : ',UID);
  Writeln (f,'GID : ',GID);
  Close (F);
  if fpChown ('test.txt',UID,GID)<>0 then
    if fpgeterrno=ESysEPERM then
      Writeln ('You are not root !')
    else
      Writeln ('Chmod failed with exit code : ',fpgeterrno)
  else
    Writeln ('Changed owner successfully !');
end.
```

1.4.7 FpClose

Synopsis: Close file descriptor.

Declaration: `function FpClose(fd: cint) : cint`

Visibility: default

Description: `FpClose` closes a file with file descriptor `Fd`. The function returns zero if the file was closed successfully, a nonzero return value indicates an error.

For an example, see FpOpen (211).

Errors: Extended error information can be retrieved using fpGetErrno (198).

See also: FpOpen (211), FpRead (215), FpWrite (236), FpFTruncate (196), FpLSeek (204)

1.4.8 FpClosedir

Synopsis: Close directory file descriptor.

Declaration:
```
function FpClosedir(var dirp: Dir) : cint
```

Visibility: default

Description: `FpCloseDir` closes the directory pointed to by `dirp`. It returns zero if the directory was closed successfully, -1 otherwise.

For an example, see fpOpenDir (212).

Errors: Extended error information can be retrieved using fpGetErrno (198).

See also: FpOpenDir (212), FpReadDir (216)

1.4.9 FpDup

Synopsis: Duplicate a file handle.

Declaration:
```
function FpDup(fildes: cint) : cint
function FpDup(var oldfile: text; var newfile: text) : cint
function FpDup(var oldfile: File; var newfile: File) : cint
```

Visibility: default

Description: `FpDup` returns a file descriptor that is a duplicate of the file descriptor `fildes`.

The second and third forms make `NewFile` an exact copy of `OldFile`, after having flushed the buffer of `OldFile` in case it is a Text file or untyped file. Due to the buffering mechanism of Pascal, these calls do not have the same functionality as the `dup` call in C. The internal Pascal buffers are not the same after this call, but when the buffers are flushed (e.g. after output), the output is sent to the same file. Doing an lseek will, however, work as in C, i.e. doing a lseek will change the file position in both files.

The function returns a negative value in case of an error, a positive value is a file handle, and indicates success.

Errors: A negative value can be one of the following error codes:

sys_ebadf`OldFile` hasn't been assigned.

sys_emfile Maximum number of open files for the process is reached.

See also: fpDup2 (190)

Listing: ./examples/bunixex/ex31.pp

```
program Example31;

{ Program to demonstrate the Dup function. }

uses baseunix;

var f : text;

begin
  if fpdup (output,f)=-1 then
    WriteIn ('Dup Failed !');
  writeln ('This is written to stdout.');
```

```
  writeln (f,'This is written to the dup file, and flushed');flush(f);
  writeln
end.
```

1.4.10 FpDup2

Synopsis: Duplicate one file handle to another.

Declaration:
```
function FpDup2(fildes: cint; fildes2: cint) : cint
function FpDup2(var oldfile: text; var newfile: text) : cint
function FpDup2(var oldfile: File; var newfile: File) : cint
```

Visibility: default

Description: Makes `fildes2` or `NewFile` an exact copy of `fildes` or `OldFile`, after having flushed the buffer of `OldFile` in the case of text or untyped files.

After a call to fdup2, the 2 file descriptors point to the same physical device (a file, socket, or a terminal).

`NewFile` can be an assigned file. If `newfile` or `fildes` was open, it is closed first. Due to the buffering mechanism of Pascal, this has not the same functionality as the `dup2` call in C. The internal Pascal buffers are not the same after this call, but when the buffers are flushed (e.g. after output), the output is sent to the same file. Doing an lseek will, however, work as in C, i.e. doing a lseek will change the file position in both files.

The function returns the new file descriptor number, on error -1 is returned, and the error can be retrieved with fpgeterrno (198)

Errors: In case of error, the following error codes can be reported:

sys_ebadf`OldFile` (or `fildes`) hasn't been assigned.

sys_emfile Maximum number of open files for the process is reached.

See also: fpDup (189)

Listing: ./examples/bunixex/ex32.pp

```
program Example32;

{ Program to demonstrate the FpDup2 function. }

uses BaseUnix;

var f : text;
    i : longint;

begin
  Assign (f,'text.txt');
  Rewrite (F);
  For i:=1 to 10 do writeln (F,'Line : ',i);
  if fpdup2 (output,f)=-1 then
    WriteIn ('Dup2 Failed !');
  writeln ('This is written to stdout.');
  writeln (f,'This is written to the dup file, and flushed');
  flush(f);
  writeln;
  { Remove file. Comment this if you want to check flushing.}
```

```
  fpUnlink ('text.txt');
end.
```

1.4.11 FpExecv

Synopsis: Execute process.

Declaration:
```
function FpExecv(path: PChar; argv: PPChar) : cint
function FpExecv(const path: RawByteString; argv: PPChar) : cint
```

Visibility: default

Description: Replaces the currently running program with the program, specified in `path`. It gives the program the options in `argvp`. This is a pointer to an array of pointers to null-terminated strings. The last pointer in this array should be nil. The current environment is passed to the program. On success, `execv` does not return.

Errors: On error, -1 is returned. Extended error information can be retrieved with fpGetErrNo (198)

sys_eacces File is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_eperm The file system is mounted .

sys_e2big Argument list too big.

sys_enoexec The magic number in the file is incorrect.

sys_enoent The file does not exist.

sys_enomem Not enough memory for kernel.

sys_enotdirA component of the path is not a directory.

sys_eloop The path contains a circular reference (via symlinks).

See also: fpExecve (192), fpFork (195)

Listing: ./examples/bunixex/ex8.pp

```
Program Example8;

{ Program to demonstrate the Execv function. }

Uses Unix, strings;

Const Arg0 : PChar = '/bin/ls';
      Arg1 : Pchar = '-l';

Var PP : PPchar;


begin
  GetMem (PP,3*SizeOf(Pchar));
  PP[0]:=Arg0;
  PP[1]:=Arg1;
  PP[3]:=Nil;
  { Execute '/bin/ls -l', with current environment }
  fpExecv ('/bin/ls',pp);
end.
```

1.4.12 FpExecve

Synopsis: Execute process using environment.

Declaration:
```
function FpExecve(path: PChar; argv: PPChar; envp: PPChar) : cint
function FpExecve(const path: RawByteString; argv: PPChar; envp: PPChar)
                  : cint
```

Visibility: default

Description: Replaces the currently running program with the program, specified in `path`. It gives the program the options in `argv`, and the environment in `envp`. They are pointers to an array of pointers to null-terminated strings. The last pointer in this array should be `nil`. On success, `execve` does not return.

Errors: Extended error information can be retrieved with fpGetErrno (198), and includes the following:

sys_eacces File is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_eperm The file system is mounted .

sys_e2big Argument list too big.

sys_enoexec The magic number in the file is incorrect.

sys_enoent The file does not exist.

sys_enomem Not enough memory for kernel.

sys_enotdirA component of the path is not a directory.

sys_eloop The path contains a circular reference (via symlinks).

See also: fpExecv (191), fpFork (195)

Listing: ./examples/bunixex/ex7.pp

```
Program Example7;

{ Program to demonstrate the Execve function. }

Uses BaseUnix, strings;

Const Arg0 : PChar = '/bin/ls';
      Arg1 : Pchar = '-l';

Var PP : PPchar;


begin
  GetMem (PP,3*SizeOf(Pchar));
  PP[0]:=Arg0;
  PP[1]:=Arg1;
  PP[3]:=Nil;
  { Execute '/bin/ls -l', with current environment }
  { Envp is defined in system.inc }
  fpExecVe ('/bin/ls',pp,envp);
end.
```

1.4.13 FpExit

Synopsis: Exit the current process.

Declaration:
```
procedure FpExit(Status: cint)
```

Visibility: default

Description: `FpExit` exits the currently running process, and report `Status` as the exit status.

Remark If this call is executed, the normal unit finalization code will not be executed. This may lead to unexpected errors and stray files on your system. It is therefore recommended to use the `Halt` call instead.

Errors: None.

See also: FpFork (195), FpExecve (192)

1.4.14 FpFcntl

Synopsis: File control operations.

Declaration:
```
function FpFcntl(fildes: cint; cmd: cint) : cint
function FpFcntl(fildes: cint; cmd: cint; arg: cint) : cint
function FpFcntl(fildes: cint; cmd: cint; var arg: FLock) : cint
```

Visibility: default

Description: Read/set a file's attributes. `Fildes` a valid file descriptor. `Cmd` specifies what to do, and is one of the following:

F_GetFd Read the close_on_exec flag. If the low-order bit is 0, then the file will remain open across execve calls.

F_GetFl Read the descriptor's flags.

F_GetOwn Get the Process ID of the owner of a socket.

F_SetFd Set the close_on_exec flag of `fildes`. (only the least significant bit is used).

F_GetLk Return the `flock` record that prevents this process from obtaining the lock, or set the `l_type` field of the lock of there is no obstruction. `Arg` is the flock record.

F_SetLk Set the lock or clear it (depending on `l_type` in the `flock` structure). if the lock is held by another process, an error occurs.

F_GetLkw Same as for **F_Setlk**, but wait until the lock is released.

F_SetOwn Set the Process or process group that owns a socket.

The function returns 0 if successful, -1 otherwise.

Errors: On error, -1 is returned. Use fpGetErrno (198) for extended error information.

sys_ebadf `Fd` has a bad file descriptor.

sys_eagain or sys_eaccess For , if the lock is held by another process.

1.4.15 fpfdfillset

Synopsis: Set all file descriptors in the set.

Declaration: `function fpfdfillset(var nset: TFDSet) : cint`

Visibility: default

Description: `fpfdfillset` sets all file descriptors in `nset`.

See also: FpSelect (219), FpFD_ZERO (195), FpFD_IsSet (194), FpFD_Clr (194), FpFD_Set (194)

1.4.16 fpFD_CLR

Synopsis: Clears a file descriptor in a set.

Declaration: `function fpFD_CLR(fdno: cint; var nset: TFDSet) : cint`

Visibility: default

Description: `FpFD_Clr` clears file descriptor `fdno` in file descriptor set `nset`.

For an example, see FpSelect (219).

Errors: None.

See also: FpSelect (219), FpFD_ZERO (195), FpFD_Set (194), FpFD_IsSet (194)

1.4.17 fpFD_ISSET

Synopsis: Check whether a file descriptor is set.

Declaration: `function fpFD_ISSET(fdno: cint; const nset: TFDSet) : cint`

Visibility: default

Description: `FpFD_Set` Checks whether file descriptor `fdNo` in file descriptor set `fds` is set. It returns zero if the descriptor is not set, 1 if it is set. If the number of the file descriptor it wrong, -1 is returned.

For an example, see FpSelect (219).

Errors: If an invalid file descriptor number is passed, -1 is returned.

See also: FpSelect (219), FpFD_ZERO (195), FpFD_Clr (194), FpFD_Set (194)

1.4.18 fpFD_SET

Synopsis: Set a file descriptor in a set.

Declaration: `function fpFD_SET(fdno: cint; var nset: TFDSet) : cint`

Visibility: default

Description: `FpFD_Set` sets file descriptor `fdno` in file descriptor set `nset`.

For an example, see FpSelect (219).

Errors: None.

See also: FpSelect (219), FpFD_ZERO (195), FpFD_Clr (194), FpFD_IsSet (194)

1.4.19 fpFD_ZERO

Synopsis: Clear all file descriptors in set.

Declaration:
```
function fpFD_ZERO(out nset: TFDSet) : cint
```

Visibility: default

Description: `FpFD_ZERO` clears all the file descriptors in the file descriptor set `nset`.

For an example, see FpSelect (219).

Errors: None.

See also: FpSelect (219), FpFD_Clr (194), FpFD_Set (194), FpFD_IsSet (194)

1.4.20 FpFork

Synopsis: Create child process.

Declaration:
```
function FpFork : TPid
```

Visibility: default

Description: `FpFork` creates a child process which is a copy of the parent process. `FpFork` returns the process ID in the parent process, and zero in the child's process. (you can get the parent's PID with fpGetPPid (200)).

Errors: On error, -1 is returned to the parent, and no child is created.

sys_eagain Not enough memory to create child process.

See also: fpExecve (192), #rtl.linux.Clone (994)

1.4.21 FPFStat

Synopsis: Retrieve file information about a file descriptor.

Declaration:
```
function FpFStat(fd: cint; var sb: Stat) : cint
function FPFStat(var F: Text; var Info: Stat) : Boolean
function FPFStat(var F: File; var Info: Stat) : Boolean
```

Visibility: default

Description: `FpFStat` gets information about the file specified in one of the following:

Fda valid file descriptor.

Fan opened text file or untyped file.

and stores it in `Info`, which is of type stat (240). The function returns zero if the call was successful, a nonzero return value indicates failure.

Errors: Extended error information can be retrieved using fpGetErrno (198).

sys_enoent`Path` does not exist.

See also: FpStat (228), FpLStat (205)

Listing: ./examples/bunixex/ex28.pp

```
program example28;

{ Program to demonstrate the FStat function. }

uses BaseUnix;

var f : text;
    i : byte;
    info : stat;

begin
  { Make a file }
  assign (f,'test.fil');
  rewrite (f);
  for i:=1 to 10 do writeln (f,'Testline # ',i);
  close (f);
  { Do the call on made file. }
  if fpstat ('test.fil',info)<>0 then
     begin
       writeln('Fstat failed. Errno : ',fpgeterrno);
       halt (1);
     end;
  writeln;
  writeln ('Result of fstat on file ''test.fil''.');
  writeln ('Inode   : ',info.st_ino);
  writeln ('Mode    : ',info.st_mode);
  writeln ('nlink   : ',info.st_nlink);
  writeln ('uid     : ',info.st_uid);
  writeln ('gid     : ',info.st_gid);
  writeln ('rdev    : ',info.st_rdev);
  writeln ('Size    : ',info.st_size);
  writeln ('Blksize : ',info.st_blksize);
  writeln ('Blocks  : ',info.st_blocks);
  writeln ('atime   : ',info.st_atime);
  writeln ('mtime   : ',info.st_mtime);
  writeln ('ctime   : ',info.st_ctime);
  { Remove file }
  erase (f);
end.
```

1.4.22 FpFtruncate

Synopsis: Truncate file on certain size.

Declaration: `function FpFtruncate(fd: cint; flength: TOff) : cint`

Visibility: default

Description: `FpFTruncate` sets the length of a file in `fd` on `flength` bytes, where `flength` must be less than or equal to the current length of the file in `fd`.

The function returns zero if the call was successful, a nonzero return value indicates that an error occurred.

Errors: Extended error information can be retrieved using fpGetErrno (198).

See also: FpOpen (211), FpClose (188), FpRead (215), FpWrite (236), FpLSeek (204)

1.4.23 FpGetcwd

Synopsis: Retrieve the current working directory.

Declaration:
```
function FpGetcwd(path: PChar; siz: TSize) : PChar
function FpGetcwd : RawByteString
```

Visibility: default

Description: `fpgetCWD` returns the current working directory of the running process. It is returned in `Path`, which points to a memory location of at least `siz` bytes.

If the function is successful, a pointer to `Path` is returned, or a string with the result. On error `Nil` or an empty string are returned.

Errors: On error `Nil` or an empty string are returned.

See also: FpGetPID (200), FpGetUID (202)

1.4.24 FpGetegid

Synopsis: Return effective group ID.

Declaration:
```
function FpGetegid : TGid
```

Visibility: default

Description: `FpGetegid` returns the effective group ID of the currently running process.

Errors: None.

See also: FpGetGid (199), FpGetUid (202), FpGetEUid (198), FpGetPid (200), FpGetPPid (200), fpSetUID (222), FpSetGid (221)

Listing: ./examples/bunixex/ex18.pp

```
Program Example18;

{ Program to demonstrate the GetGid and GetEGid functions. }

Uses BaseUnix;

begin
 writeln ('Group Id = ',fpgetgid,' Effective group Id = ',fpgetegid);
end.
```

1.4.25 FpGetEnv

Synopsis: Return value of environment variable.

Declaration:
```
function FpGetEnv(name: PChar) : PChar
function FpGetEnv(name: string) : PChar
```

Visibility: default

Description: `FPGetEnv` returns the value of the environment variable in `Name`. If the variable is not defined, nil is returned. The value of the environment variable may be the empty string. A PChar is returned to accommodate for strings longer than 255 bytes, `TERMCAP` and `LS_COLORS`, for instance.

Errors: None.

Listing: ./examples/bunixex/ex41.pp

```
Program Example41;

{ Program to demonstrate the GetEnv function. }

Uses BaseUnix;

begin
  WriteIn ('Path is : ',fpGetenv('PATH'));
end.
```

1.4.26 fpgeterrno

Synopsis: Retrieve extended error information.

Declaration: `function fpgeterrno : LongInt`

Visibility: default

Description: `fpgeterrno` returns extended information on the latest error. It is set by all functions that communicate with the kernel or C library.

Errors: None.

See also: fpseterrno (220)

1.4.27 FpGeteuid

Synopsis: Return effective user ID.

Declaration: `function FpGeteuid : TUid`

Visibility: default

Description: `FpGeteuid` returns the effective user ID of the currently running process.

Errors: None.

See also: FpGetUid (202), FpGetGid (199), FpGetEGid (197), FpGetPid (200), FpGetPPid (200), fpSetUID (222), FpSetGid (221)

Listing: ./examples/bunixex/ex17.pp

```
Program Example17;

{ Program to demonstrate the GetUid and GetEUid functions. }

Uses BaseUnix;

begin
  writeln ('User Id = ',fpgetuid,' Effective user Id = ',fpgeteuid);
end.
```

1.4.28 FpGetgid

Synopsis: Return real group ID.

Declaration: `function FpGetgid : TGid`

Visibility: default

Description: `FpGetgid` returns the real group ID of the currently running process.

Errors: None.

See also: FpGetEGid (197), FpGetUid (202), FpGetEUid (198), FpGetPid (200), FpGetPPid (200), fpSetUID (222), FpSetGid (221)

Listing: ./examples/bunixex/ex18.pp

```
Program Example18;

{ Program to demonstrate the GetGid and GetEGid functions. }

Uses BaseUnix;

begin
 writeln ('Group Id = ',fpgetgid,' Effective group Id = ',fpgetegid);
end.
```

1.4.29 FpGetgroups

Synopsis: Get the list of supplementary groups.

Declaration: `function FpGetgroups(gidsetsize: cint; var grouplist: TGrpArr) : cint`

Visibility: default

Description: `FpGetgroups` returns up to `gidsetsize` groups in `GroupList`

If the function is successful, then number of groups that were stored is returned. On error, -1 is returned.

Errors: On error, -1 is returned. Extended error information can be retrieved with fpGetErrNo (198)

See also: FpGetpgrp (199), FpGetGID (199), FpGetEGID (197)

1.4.30 FpGetpgrp

Synopsis: Get process group ID.

Declaration: `function FpGetpgrp : TPid`

Visibility: default

Description: `FpGetpgrp` returns the process group ID of the current process.

Errors: None.

See also: fpGetPID (200), fpGetPPID (200), FpGetGID (199), FpGetUID (202)

1.4.31 FpGetpid

Synopsis: Return current process ID.

Declaration: `function FpGetpid : TPid`

Visibility: default

Description: `FpGetpid` returns the process ID of the currently running process.

Note: There exist a portable alternative to fpGetpid: system.GetProcessID. Please use fpGetpid only if you are writing Unix specific code. System.GetProcessID will work on all operating systems.

Errors: None.

See also: FpGetPPid (200)

Listing: ./examples/bunixex/ex16.pp

```
Program Example16;

{ Program to demonstrate the GetPid, GetPPid function. }

Uses BaseUnix;

begin
  WriteIn ('Process Id = ',fpgetpid,' Parent process Id = ',fpgetppid);
end.
```

1.4.32 FpGetppid

Synopsis: Return parent process ID.

Declaration: `function FpGetppid : TPid`

Visibility: default

Description: `FpGetppid` returns the Process ID of the parent process.

Errors: None.

See also: FpGetPid (200)

Listing: ./examples/bunixex/ex16.pp

```
Program Example16;

{ Program to demonstrate the GetPid, GetPPid function. }

Uses BaseUnix;

begin
  WriteIn ('Process Id = ',fpgetpid,' Parent process Id = ',fpgetppid);
end.
```

1.4.33 fpGetPriority

Synopsis: Return process priority.

Declaration: `function fpGetPriority(Which: cint; Who: cint) : cint`

Visibility: default

Description: GetPriority returns the priority with which a process is running. Which process(es) is determined by the `Which` and `Who` variables. `Which` can be one of the predefined `Prio_Process`, `Prio_PGrp`, `Prio_User`, in which case `Who` is the process ID, Process group ID or User ID, respectively.

For an example, see FpNice (210).

Errors: Error information is returned solely by the FpGetErrno (198) function: a priority can be a positive or negative value.

sys_esrch No process found using `which` and `who`.

sys_einval`Which` was not one of `Prio_Process`, `Prio_Grp` or `Prio_User`.

See also: FpSetPriority (221), FpNice (210)

1.4.34 FpGetRLimit

Synopsis: Get process resource limits.

Declaration: `function FpGetRLimit(resource: cint; rlim: PRLimit) : cint`

Visibility: default

Description: `FpGetRLimit` gets the resource limits for the current process: `resource` determines the resource of which the kernel should return the limits (one of the many `RLIMIT_*` constants). `rlim` should point to a TRLimit (243) record and on success will contain the resource limits.

The function returns zero if the resource limits were correctly returned.

Errors: On error, -1 is returned and fpgeterrno (198) can be used to retrieve the error code.

See also: FpSetRLimit (221)

1.4.35 FpGetsid

Synopsis: Get current session ID.

Declaration: `function FpGetsid(pid: TPid) : TPid`

Visibility: default

Description: `FpGetsid` returns the session ID of the process `pid`. The return value is the session ID of the process. (it equals the PID of the session leader). The process pid must be in the same session as the current process.

Errors: On error, -1 is returned, and extended error information can be obtained with `fpGetErrno`.

See also: FpGetpgrp (199), FpGetpid (200), FpGetPpid (200)

1.4.36 FpGetuid

Synopsis: Return current user ID.

Declaration: `function FpGetuid : TUid`

Visibility: default

Description: `FpGetuid` returns the real user ID of the currently running process.

Errors: None.

See also: FpGetGid (199), FpGetEUid (198), FpGetEGid (197), FpGetPid (200), FpGetPPid (200), fpSetUID (222)

Listing: ./examples/bunixex/ex17.pp

```
Program Example17;

{ Program to demonstrate the GetUid and GetEUid functions. }

Uses BaseUnix;

begin
  writeln ('User Id = ',fpgetuid,' Effective user Id = ',fpgeteuid);
end.
```

1.4.37 FpIOCtl

Synopsis: General kernel IOCTL call.

Declaration: `function FpIOCtl(Handle: cint; Ndx: TIOCtlRequest; Data: Pointer) : cint`

Visibility: default

Description: This is a general interface to the Unix/ Linux ioctl call. It performs various operations on the file descriptor `Handle`. `Ndx` describes the operation to perform. `Data` points to data needed for the `Ndx` function. The structure of this data is function-dependent, so we don't elaborate on this here. For more information on this, see various manual pages under Linux.

Errors: Extended error information can be retrieved using fpGetErrno (198).

Listing: ./examples/bunixex/ex54.pp

```
Program Example54;

uses BaseUnix,Termio;

{ Program to demonstrate the IOCtl function. }

var
  tios : Termios;

begin
  {$ifdef FreeBSD}
    fpIOCtl(1,TIOCGETA,@tios);   // these constants are very OS dependant.
                                 // see the tcgetattr example for a better way
  {$endif}
  WriteLn('Input Flags  : $',hexstr(tios.c_iflag,8));
```

```
  WriteLn('Output Flags : $',hexstr(tios.c_oflag,8));
  WriteLn('Line Flags   : $',hexstr(tios.c_lflag,8));
  WriteLn('Control Flags: $',hexstr(tios.c_cflag,8));
end.
```

1.4.38 FpKill

Synopsis: Send a signal to a process.

Declaration:
```
function FpKill(pid: TPid; sig: cint) : cint
```

Visibility: default

Description: `fpKill` sends a signal `Sig` to a process or process group. If `Pid`>0 then the signal is sent to `Pid`, if it equals -1, then the signal is sent to all processes except process 1. If `Pid`<-1 then the signal is sent to process group -Pid.

The return value is zero, except in case three, where the return value is the number of processes to which the signal was sent.

Errors: Extended error information can be retrieved using fpGetErrno (198):

sys_einval An invalid signal is sent.

sys_esrch The `Pid` or process group don't exist.

sys_eperm The effective userid of the current process doesn't math the one of process `Pid`.

See also: FpSigAction (223), FpSignal (225)

1.4.39 FpLink

Synopsis: Create a hard link to a file.

Declaration:
```
function FpLink(existing: PChar; newone: PChar) : cint
function FpLink(const existing: RawByteString;
                const newone: RawByteString) : cint
```

Visibility: default

Description: `fpLink` makes `NewOne` point to the same file as `Existing`. The two files then have the same inode number. This is known as a 'hard' link. The function returns `zero` if the call was successful, and returns a `non-zero` value if the call failed.

Errors: The following error codes are returned:

sys_exdev `Existing` and `NewOne` are not on the same file system.

sys_eperm The file system containing `Existing` and `NewOne` doesn't support linking files.

sys_eaccess Write access for the directory containing `NewOne` is disallowed, or one of the directories in `Existing` or `NewOne` has no search (=execute) permission.

sys_enoent A directory entry in `Existing` or `NewOne` does not exist or is a symbolic link pointing to a non-existent directory.

sys_enotdir A directory entry in `Existing` or `NewOne` is nor a directory.

sys_enomem Insufficient kernel memory.

sys_erofs The files are on a read-only file system.

sys_eexist`NewOne` already exists.

sys_emlink`Existing` has reached maximal link count.

sys_eloop`existing` or `NewOne` has a reference to a circular symbolic link, i.e. a symbolic link, whose expansion points to itself.

sys_enospc The device containing `NewOne` has no room for another entry.

sys_eperm`Existing` points to . or .. of a directory.

See also: fpSymLink (229), fpUnLink (234)

Listing: ./examples/bunixex/ex21.pp

```
Program Example21;

{ Program to demonstrate the Link and UnLink functions. }

Uses BaseUnix;

Var F : Text;
    S : String;
begin
  Assign (F,'test.txt');
  Rewrite (F);
  Writeln (F,'This is written to test.txt');
  Close(f);
  { new.txt and test.txt are now the same file }
  if fpLink ('test.txt','new.txt')<>0 then
    writeln ('Error when linking !');
  { Removing test.txt still leaves new.txt }
  If fpUnlink ('test.txt')<>0 then
    Writeln ('Error when unlinking !');
  Assign (f,'new.txt');
  Reset (F);
  While not EOF(f) do
    begin
    Readln(F,S);
    Writeln ('> ',s);
    end;
 Close (f);
 { Remove new.txt also }
 If not FPUnlink ('new.txt')<>0 then
   Writeln ('Error when unlinking !');
end.
```

1.4.40 FpLseek

Synopsis: Set file pointer position.

Declaration: `function FpLseek(fd: cint; offset: TOff; whence: cint) : TOff`

Visibility: default

Description: `FpLSeek` sets the current file position of file `fd` to `Offset`, starting from `Whence`, which can be one of the following:

Seek_Set`Offset` is the absolute position in the file.

Seek_Cur`Offset` is relative to the current position.

Seek_end`Offset` is relative to the end of the file.

The function returns the new file position, or -1 of an error occurred.

For an example, see FpOpen (211).

Errors: Extended error information can be retrieved using fpGetErrno (198).

See also: FpOpen (211), FpWrite (236), FpClose (188), FpRead (215), FpFTruncate (196)

1.4.41 fpLstat

Synopsis: Return information about symbolic link. Do not follow the link.

Declaration:
```
function fpLstat(path: PChar; Info: PStat) : cint
function fpLstat(const path: RawByteString; Info: PStat) : cint
function fpLstat(path: PChar; var Info: Stat) : cint
function fpLstat(const Filename: RawByteString; var Info: Stat) : cint
```

Visibility: default

Description: `FpLstat` gets information about the link specified in `Path` (or `FileName`, and stores it in `Info`, which points to a record of type `TStat`. Contrary to FpFstat (195), it stores information about the link, not about the file the link points to. The function returns zero if the call was successful, a nonzero return value indicates failure. failed.

Errors: Extended error information is returned by the FpGetErrno (198) function.

sys_enoent`Path` does not exist.

See also: FpFStat (195), #rtl.unixtype.TStatFS (2189)

Listing: ./examples/unixex/ex29.pp

```
program example29;

{ Program to demonstrate the LStat function. }

uses BaseUnix,Unix;

var f : text;
    i : byte;
    info : stat;

begin
  { Make a file }
  assign (f,'test.fil');
  rewrite (f);
  for i:=1 to 10 do writeln (f,'Testline # ',i);
  close (f);
  { Do the call on made file. }
  if fpstat ('test.fil',info)<>0 then
     begin
     writeln('Fstat failed. Errno : ',fpgeterrno);
     halt (1);
     end;
  writeln;
```

```
  writeln ('Result of stat on file ''test.fil''.');
  writeln ('Inode    : ',info.st_ino);
  writeln ('Mode     : ',info.st_mode);
  writeln ('nlink    : ',info.st_nlink);
  writeln ('uid      : ',info.st_uid);
  writeln ('gid      : ',info.st_gid);
  writeln ('rdev     : ',info.st_rdev);
  writeln ('Size     : ',info.st_size);
  writeln ('Blksize  : ',info.st_blksize);
  writeln ('Blocks   : ',info.st_blocks);
  writeln ('atime    : ',info.st_atime);
  writeln ('mtime    : ',info.st_mtime);
  writeln ('ctime    : ',info.st_ctime);

  If  fpSymLink ('test.fil','test.lnk')<>0 then
    writeln ('Link failed ! Errno :',fpgeterrno);

  if  fplstat ('test.lnk',@info)<>0 then
     begin
     writeln('LStat failed. Errno : ',fpgeterrno);
     halt (1);
     end;
  writeln;
  writeln ('Result of fstat on file ''test.lnk''.');
  writeln ('Inode    : ',info.st_ino);
  writeln ('Mode     : ',info.st_mode);
  writeln ('nlink    : ',info.st_nlink);
  writeln ('uid      : ',info.st_uid);
  writeln ('gid      : ',info.st_gid);
  writeln ('rdev     : ',info.st_rdev);
  writeln ('Size     : ',info.st_size);
  writeln ('Blksize  : ',info.st_blksize);
  writeln ('Blocks   : ',info.st_blocks);
  writeln ('atime    : ',info.st_atime);
  writeln ('mtime    : ',info.st_mtime);
  writeln ('ctime    : ',info.st_ctime);
  { Remove file and link }
  erase (f);
  fpunlink ('test.lnk');
end.
```

1.4.42 FpMkdir

Synopsis: Create a new directory.

Declaration:
```
function FpMkdir(path: PChar; Mode: TMode) : cint
function FpMkdir(const path: RawByteString; Mode: TMode) : cint
```

Visibility: default

Description: `FpMkDir` creates a new directory `Path`, and sets the new directory's mode to `Mode`. Path can be an absolute path or a relative path. Note that only the last element of the directory will be created, higher level directories must already exist, and must be writeable by the current user.

On success, 0 is returned. if the function fails, -1 is returned.

Note: There exist a portable alternative to fpMkDir: system.mkdir. Please use fpMkDir only if you are writing Unix specific code. System.mkdir will work on all operating systems.

Errors: Extended error information can be retrieved using fpGetErrno (198).

See also: fpGetCWD (197), fpChDir (186)

1.4.43 FpMkfifo

Synopsis: Create FIFO (named pipe) in file system.

Declaration:
```
function FpMkfifo(path: PChar; Mode: TMode) : cint
function FpMkfifo(const path: RawByteString; Mode: TMode) : cint
```

Visibility: default

Description: `fpMkFifo` creates named a named pipe in the file system, with name `Path` and mode `Mode`.

The function returns zero if the command was successful, and nonzero if it failed.

Errors: The error codes include:

sys_emfile Too many file descriptors for this process.

sys_enfile The system file table is full.

1.4.44 Fpmmap

Synopsis: Create memory map of a file.

Declaration:
```
function Fpmmap(start: pointer; len: size_t; prot: cint; flags: cint;
                fd: cint; offst: off_t) : pointer
```

Visibility: default

Description: `FpMMap` maps or unmaps files or devices into memory. The different arguments determine what and how the file is mapped:

adr Address where to mmap the device. This address is a hint, and may not be followed.

len Size (in bytes) of area to be mapped.

prot Protection of mapped memory. This is a OR-ed combination of the following constants:

- **PROT_EXEC** The memory can be executed.
- **PROT_READ** The memory can be read.
- **PROT_WRITE** The memory can be written.
- **PROT_NONE** The memory can not be accessed.

flags Contains some options for the mmap call. It is an OR-ed combination of the following constants:

- **MAP_FIXED** Do not map at another address than the given address. If the address cannot be used, `MMap` will fail.
- **MAP_SHARED** Share this map with other processes that map this object.
- **MAP_PRIVATE** Create a private map with copy-on-write semantics.
- **MAP_ANONYMOUS** `fd` does not have to be a file descriptor.

One of the options `MAP_SHARED` and `MAP_PRIVATE` must be present, but not both at the same time.

fd File descriptor from which to map.

off Offset to be used in file descriptor `fd`.

The function returns a pointer to the mapped memory, or a -1 in case of en error.

Errors: On error, -1 is returned and extended error information is returned by the FpGetErrno (198) function.

Sys_EBADF `fd` is not a valid file descriptor and `MAP_ANONYMOUS` was not specified.

Sys_EACCES `MAP_PRIVATE` was specified, but `fd` is not open for reading. Or `MAP_SHARED` was asked and `PROT_WRITE` is set, fd is not open for writing

Sys_EINVAL One of the record fields `Start`, `length` or `offset` is invalid.

Sys_ETXTBUSY `MAP_DENYWRITE` was set but the object specified by fd is open for writing.

Sys_EAGAIN `fd` is locked, or too much memory is locked.

Sys_ENOMEM Not enough memory for this operation.

See also: FpMUnMap (209)

Listing: ./examples/unixex/ex66.pp

```
Program Example66;

{ Program to demonstrate the MMap function. }

Uses BaseUnix,Unix;

Var S    : String;
    fd   : cint;
    Len  : longint;
//    args : tmmapargs;
    P    : PChar;

begin
  s:='This is the string';
  Len:=Length(S);
  fd:=fpOpen('testfile.txt',O_wrOnly or o_creat);
  If fd=-1 then
    Halt(1);
  If fpWrite(fd,S[1],Len)=-1 then
    Halt(2);
  fpClose(fd);
  fd:=fpOpen('testfile.txt',O_rdOnly);
  if fd=-1 then
    Halt(3);
  P:=Pchar(fpmmap(nil,len+1 ,PROT_READ or PROT_WRITE,MAP_PRIVATE,fd,0));

  If longint(P)=-1 then
    Halt(4);
  Writeln('Read in memory  :',P);
  fpclose(fd);
  if fpMUnMap(P,Len)<>0 Then
    Halt(fpgeterrno);
end.
```

1.4.45 Fpmprotect

Synopsis: Set memory protection.

Declaration: `function Fpmprotect(start: pointer; len: size_t; prot: cint) : cint`

Visibility: default

Description: `fpmprotect` sets memory protection on a certain block of memory. The memory block to protect starts at `adr` and has size `len`. The `prot` argument specifies the kind of protection to apply, and is a bit-wise OR-ed combination of the following:

PROT_NONE Memory cannot be accessed

PROT_READ Memory can be read

PROT_WRITE Memory can be written

PROT_EXEC Memory can be executed

PROT_SEM Memory can be used for atomic operations

PROT_SAO Memory must have strong access order (powerPC only).

1.4.46 Fpmunmap

Synopsis: Unmap previously mapped memory block.

Declaration: `function Fpmunmap(start: pointer; len: size_t) : cint`

Visibility: default

Description: `FpMUnMap` unmaps the memory block of size `Len`, pointed to by `Adr`, which was previously allocated with FpMMap (207).

The function returns `True` if successful, `False` otherwise.

For an example, see FpMMap (207).

Errors: In case of error the function returns a nonzero value, extended error information is returned by the FpGetErrno (198) function. See FpMMap (207) for possible error values.

See also: FpMMap (207)

1.4.47 FpNanoSleep

Synopsis: Suspend process for a short time.

Declaration: `function FpNanoSleep(req: ptimespec; rem: ptimespec) : cint`

Visibility: default

Description: `FpNanoSleep` suspends the process till a time period as specified in `req` has passed. Then the function returns. If the call was interrupted (e.g. by some signal) then the function may return earlier, and `rem` will contain the remaining time till the end of the intended period. In this case the return value will be -1, and `ErrNo` will be set to `EINTR`

If the function returns without error, the return value is zero.

Errors: If an error occurred or the call was interrupted, -1 is returned. Extended error information can be retrieved using fpGetErrno (198).

See also: FpPause (213), FpAlarm (185)

Listing: ./examples/bunixex/ex72.pp

```
program example72;

{ Program to demonstrate the NanoSleep function. }

uses BaseUnix;

Var
  Req,Rem : TimeSpec;
  Res : Longint;

begin
  With Req do
    begin
    tv_sec:=10;
    tv_nsec:=100;
    end;
  Write('NanoSleep returned : ');
  Flush(Output);
  Res:=(fpNanoSleep(@Req,@rem));
  Writeln(res);
  If (res<>0) then
    With rem do
      begin
      Writeln('Remaining seconds      : ',tv_sec);
      Writeln('Remaining nanoseconds : ',tv_nsec);
      end;
end.
```

1.4.48 fpNice

Synopsis: Set process priority.

Declaration: `function fpNice(N: cint) : cint`

Visibility: default

Description: `Nice` adds `-N` to the priority of the running process. The lower the priority numerically, the less the process is favored. Only the superuser can specify a negative `N`, i.e. increase the rate at which the process is run.

If the function is successful, zero is returned. On error, a nonzero value is returned.

Errors: Extended error information is returned by the FpGetErrno (198) function.

sys_eperm A non-superuser tried to specify a negative `N`, i.e. do a priority increase.

See also: FpGetPriority (201), FpSetPriority (221)

Listing: ./examples/unixex/ex15.pp

```
Program Example15;

{ Program to demonstrate the Nice and Get/SetPriority functions. }

Uses BaseUnix, Unix;

begin
```

```
  writeln ('Setting priority to 5');
  fpsetpriority (prio_process,fpgetpid,5);
  writeln ('New priority = ',fpgetpriority (prio_process,fpgetpid));
  writeln ('Doing nice 10');
  fpnice (10);
  writeln ('New Priority = ',fpgetpriority (prio_process,fpgetpid));
end.
```

1.4.49 FpOpen

Synopsis: Open file and return file descriptor.

Declaration:
```
function FpOpen(path: PChar; flags: cint; Mode: TMode) : cint
function FpOpen(path: PChar; flags: cint) : cint
function FpOpen(const path: RawByteString; flags: cint) : cint
function FpOpen(const path: RawByteString; flags: cint; Mode: TMode)
                : cint
function FpOpen(path: ShortString; flags: cint) : cint
function FpOpen(path: ShortString; flags: cint; Mode: TMode) : cint
```

Visibility: default

Description: `FpOpen` opens a file in `Path` with flags `flags` and mode `Mode` One of the following:

O_RdOnly File is opened Read-only

O_WrOnly File is opened Write-only

O_RdWr File is opened Read-Write

The flags may be`OR`-ed with one of the following constants:

O_Creat File is created if it doesn't exist.

O_Excl If the file is opened with `O_Creat` and it already exists, the call will fail.

O_NoCtty If the file is a terminal device, it will NOT become the process' controlling terminal.

O_Trunc If the file exists, it will be truncated.

O_Appendthe file is opened in append mode. *Before each write*, the file pointer is positioned at the end of the file.

O_NonBlock The file is opened in non-blocking mode. No operation on the file descriptor will cause the calling process to wait till.

O_NDelay Idem as `O_NonBlock`

O_Sync The file is opened for synchronous IO. Any write operation on the file will not return until the data is physically written to disk.

O_NoFollowif the file is a symbolic link, the open fails. (Linux 2.1.126 and higher only)

O_Directoryif the file is not a directory, the open fails. (Linux 2.1.126 and higher only)

`Path` can be of type `PChar` or `String`. The optional `mode` argument specifies the permissions to set when opening the file. This is modified by the umask setting. The real permissions are `Mode and not umask`. The return value of the function is the file descriptor, or a negative value if there was an error.

Errors: Extended error information can be retrieved using fpGetErrno (198).

See also: FpClose (188), FpRead (215), FpWrite (236), FpFTruncate (196), FpLSeek (204)

Listing: ./examples/bunixex/ex19.pp

```
Program Example19;

{ Program to demonstrate the fpOpen, fpwrite and fpCLose functions. }

Uses BaseUnix;

Const Line : String[80] = 'This is easy writing !';

Var FD : Cint;

begin
  FD:=fpOpen ('Test.dat',O_WrOnly or O_Creat);
  if FD>0 then
    begin
    if length(Line)<>fpwrite (FD,Line[1],Length(Line)) then
      WriteIn ('Error when writing to file !');
    fpClose(FD);
    end;
end.
```

1.4.50 FpOpendir

Synopsis: Open a directory for reading.

Declaration:
```
function FpOpendir(dirname: PChar) : pDir
function FpOpendir(const dirname: RawByteString) : pDir
function FpOpendir(dirname: ShortString) : pDir
```

Visibility: default

Description: `FpOpenDir` opens the directory `DirName`, and returns a `pdir` pointer to a Dir (238) record, which can be used to read the directory structure. If the directory cannot be opened, `nil` is returned.

Errors: Extended error information can be retrieved using fpGetErrno (198).

See also: FpCloseDir (189), FpReadDir (216)

Listing: ./examples/bunixex/ex35.pp

```
Program Example35;

{ Program to demonstrate the
  OpenDir,ReadDir, SeekDir and TellDir functions. }

Uses BaseUnix;

Var TheDir : PDir;
    ADirent : PDirent;
    Entry : Longint;

begin
  TheDir:=fpOpenDir('./.');
  Repeat
//    Entry:=fpTellDir(TheDir);
    ADirent:=fpReadDir (TheDir^);
    If ADirent<>Nil then
```

```
      With ADirent^ do
        begin
        WriteLn ('Entry No : ',Entry);
        WriteLn ('Inode    : ',d_fileno);
//        WriteLn ('Offset   : ',d_off);
        WriteLn ('Reclen   : ',d_reclen);
        WriteLn ('Name     : ',pchar(@d_name[0]));
        end;
  Until ADirent=Nil;
  Repeat
    Write ('Entry No. you would like to see again (-1 to stop): ');
    ReadLn (Entry);
    If Entry<>-1 then
      begin
//      fpSeekDir (TheDir,Entry);              // not implemented for various platforms
      ADirent:=fpReadDir (TheDir^);
      If ADirent<>Nil then
        With ADirent^ do
          begin
          WriteLn ('Entry No : ',Entry);
          WriteLn ('Inode    : ',d_fileno);
//          WriteLn ('Offset   : ',off);
          WriteLn ('Reclen   : ',d_reclen);
          WriteLn ('Name     : ',pchar(@d_name[0]));
          end;
    end;
  Until Entry=-1;
  fpCloseDir (TheDir^);
end.
```

1.4.51 FpPause

Synopsis: Wait for a signal to arrive.

Declaration: `function FpPause : cint`

Visibility: default

Description: `FpPause` puts the process to sleep and waits until the application receives a signal. If a signal handler is installed for the received signal, the handler will be called and after that pause will return control to the process.

For an example, see fpAlarm (185).

1.4.52 FpPipe

Synopsis: Create a set of pipe file handlers.

Declaration: `function FpPipe(var fildes: TFilDes) : cint`

Visibility: default

Description: `FpPipe` creates a pipe, i.e. two file objects, one for input, one for output. The file handles are returned in the array `fildes`. The input handle is in the 0-th element of the array, the output handle is in the 1-st element.

The function returns zero if everything went successfully, a nonzero return value indicates an error.

Errors: In case the function fails, the following return values are possible:

sys_emfile Too many file descriptors for this process.

sys_enfile The system file table is full.

See also: #rtl.unix.POpen (2165), fpMkFifo (207)

Listing: ./examples/bunixex/ex36.pp

```
Program Example36;

{ Program to demonstrate the AssignPipe function. }

Uses BaseUnix,Unix;

Var pipi,pipo : Text;
    s : String;

begin
  Writeln ('Assigning Pipes.');
  If assignpipe(pipi,pipo)<>0 then
    Writeln('Error assigning pipes !',fpgeterrno);
  Writeln ('Writing to pipe, and flushing.');
  Writeln (pipo,'This is a textstring');close(pipo);
  Writeln ('Reading from pipe.');
  While not eof(pipi) do
    begin
    Readln (pipi,s);
    Writeln ('Read from pipe : ',s);
    end;
  close (pipi);
  writeln ('Closed pipes.');
  writeln
end.
```

1.4.53 FpPoll

Synopsis: Poll a file descriptor for events.

Declaration: `function FpPoll(fds: ppollfd; nfds: cuint; timeout: clong) : cint`

Visibility: default

Description: `fpPoll` waits for events on file descriptors. `fds` points to an array of `tpollfd` records, each of these records describes a file descriptor on which to wait for events. The number of file descriptors is given by `nfds`. `>timeout` specifies the maximum time (in milliseconds) to wait for events.

On timeout, the result value is 0. If an event occurred on some descriptors, then the return value is the number of descriptors on which an event (or error) occurred. The `revents` field of the `tpollfd` records will contain the events for the file descriptor it described.

See also: tpollfd (182)

1.4.54 FppRead

Synopsis: Positional read: read from file descriptor at a certain position.

Declaration:
```
function FpPRead(fd: cint; buf: PChar; nbytes: TSize; offset: TOff)
                 : TsSize
function FppRead(fd: cint; var buf; nbytes: TSize; offset: TOff)
                 : TsSize
```

Visibility: default

Description: `FpPRead` reads `nbytes` bytes from file descriptor `fd` into buffer `buf` starting at offset `offset`. `Offset` is measured from the start of the file. This function can only be used on files, not on pipes or sockets (i.e. any seekable file descriptor).

The function returns the number of bytes actually read, or -1 on error.

Errors: On error, -1 is returned.

See also: FpReadV (218), FpPWrite (215)

1.4.55 FppWrite

Synopsis: Positional write: write to file descriptor at a certain position.

Declaration:
```
function FpPWrite(fd: cint; buf: PChar; nbytes: TSize; offset: TOff)
                  : TsSize
function FppWrite(fd: cint; const buf; nbytes: TSize; offset: TOff)
                  : TsSize
```

Visibility: default

Description: `FpPWrite` writes `nbytes` bytes from buffer `buf` into file descriptor `fd` starting at offset `offset`. `Offset` is measured from the start of the file. This function can only be used on files, not on pipes or sockets (i.e. any seekable file descriptor).

The function returns the number of bytes actually written, or -1 on error.

Errors: On error, -1 is returned.

See also: FpPRead (214), FpWriteV (237)

1.4.56 FpRead

Synopsis: Read data from file descriptor.

Declaration:
```
function FpRead(fd: cint; buf: PChar; nbytes: TSize) : TsSize
function FpRead(fd: cint; var buf; nbytes: TSize) : TsSize
```

Visibility: default

Description: `FpdRead` reads at most `nbytes` bytes from the file descriptor `fd`, and stores them in `buf`.

The function returns the number of bytes actually read, or -1 if an error occurred. No checking on the length of `buf` is done.

Errors: Extended error information can be retrieved using fpGetErrno (198).

See also: FpOpen (211), FpClose (188), FpWrite (236), FpFTruncate (196), FpLSeek (204)

Listing: ./examples/bunixex/ex20.pp

```
Program Example20;

{ Program to demonstrate the fdRead and fdTruncate functions. }

Uses BaseUnix;

Const Data : string[10] = '1234567890';

Var FD : cint;
    l : longint;

begin
  FD:=fpOpen('test.dat',o_wronly or o_creat,&666);
  if fd>0 then
    begin
    { Fill file with data }
    for l:=1 to 10 do
      if fpWrite (FD,Data[1],10)<>10 then
        begin
        writeln ('Error when writing !');
        halt(1);
        end;
    fpClose(FD);
    FD:=fpOpen('test.dat',o_rdonly);
    { Read data again }
    If FD>0 then
      begin
      For l:=1 to 5 do
        if fpRead (FD,Data[1],10)<>10 then
          begin
          WriteIn ('Error when Reading !');
          Halt(2);
          end;
      fpClose(FD);
      { Truncating file at 60 bytes }
      { For truncating, file must be open or write }
      FD:=fpOpen('test.dat',o_wronly,&666);
      if FD>0 then
        begin
        if fpfTruncate(FD,60)<>0 then
           WriteIn('Error when truncating !');
        fpClose (FD);
        end;
      end;
    end;
end.
```

1.4.57 FpReaddir

Synopsis: Read entry from directory.

Declaration: `function FpReaddir(var dirp: Dir) : pDirent`

Visibility: default

Description: `FpReadDir` reads the next entry in the directory pointed to by `dirp`. It returns a `pdirent` pointer to a dirent (239) record describing the entry. If the next entry can't be read, `Nil` is returned.

For an example, see FpOpenDir (212).

Errors: Extended error information can be retrieved using fpGetErrno (198).

See also: FpCloseDir (189), FpOpenDir (212)

1.4.58 fpReadLink

Synopsis: Read destination of symbolic link.

Declaration:
```
function fpReadLink(name: PChar; linkname: PChar; maxlen: size_t) : cint
function fpReadLink(const Name: RawByteString) : RawByteString
```

Visibility: default

Description: `FpReadLink` returns the file the symbolic link `name` is pointing to. The first form of this function accepts a buffer `linkname` of length `maxlen` where the filename will be stored. It returns the actual number of characters stored in the buffer.

The second form of the function returns simply the name of the file.

Errors: On error, the first form of the function returns -1; the second one returns an empty string. Extended error information is returned by the FpGetErrno (198) function.

SYS_ENOTDIR A part of the path in `Name` is not a directory.

SYS_EINVAL maxlen is not positive, or the file is not a symbolic link.

SYS_ENAMETOOLONG A pathname, or a component of a pathname, was too long.

SYS_ENOENT the link `name` does not exist.

SYS_EACCES No permission to search a directory in the path

SYS_ELOOP Too many symbolic links were encountered in translating the pathname.

SYS_EIO An I/O error occurred while reading from the file system.

SYS_EFAULT The buffer is not part of the process's memory space.

SYS_ENOMEM Not enough kernel memory was available.

See also: FpSymLink (229)

Listing: ./examples/unixex/ex62.pp

```
Program Example62;

{ Program to demonstrate the ReadLink function. }

Uses BaseUnix,Unix;

Var F : Text;
    S : String;

begin
  Assign (F,'test.txt');
  Rewrite (F);
  Writeln (F,'This is written to test.txt');
  Close(f);
  { new.txt and test.txt are now the same file }
  if fpSymLink ('test.txt','new.txt')<>0 then
    writeln ('Error when symlinking !');
  S:=fpReadLink('new.txt');
```

```
  If S='' then
    WriteLn ('Error reading link !')
  Else
    WriteLn ('Link points to : ',S);
 { Now remove links }
 If fpUnlink ('new.txt')<>0 then
   WriteLn ('Error when unlinking !');
 If fpUnlink ('test.txt')<>0 then
   WriteLn ('Error when unlinking !');
end.
```

1.4.59 FpReadV

Synopsis: Vector read: Read into multiple buffers.

Declaration: `function FpReadV(fd: cint; const iov: piovec; iovcnt: cint) : TsSize`

Visibility: default

Description: `FpReadV` reads data from file descriptor `fd` and writes it into `iovcnt` buffers described by the tiovec (182) buffers pointed to by `iov`. It works like fpRead (215) only on multiple buffers.

Errors: On error, -1 is returned.

See also: FpWriteV (237), FpPWrite (215), FpPRead (214)

1.4.60 FpRename

Synopsis: Rename file.

Declaration:
```
function FpRename(old: PChar; newpath: PChar) : cint
function FpRename(const old: RawByteString;
                  const newpath: RawByteString) : cint
```

Visibility: default

Description: `FpRename` renames the file `Old` to `NewPath`. `NewPath` can be in a different directory than `Old`, but it cannot be on another partition (device). Any existing file on the new location will be replaced.

If the operation fails, then the `Old` file will be preserved.

The function returns zero on success, a nonzero value indicates failure.

Note: There exist a portable alternative to fpRename: system.rename. Please use fpRename only if you are writing Unix specific code. System.rename will work on all operating systems.

Errors: Extended error information can be retrieved using fpGetErrno (198).

sys_eisdir`NewPath` exists and is a directory, but `Old` is not a directory.

sys_exdev`NewPath` and `Old` are on different devices.

sys_enotempty or sys_eexist`NewPath` is an existing, non-empty directory.

sys_ebusy`Old` or `NewPath` is a directory and is in use by another process.

sys_einval`NewPath` is part of `Old`.

sys_emlink`OldPath` or `NewPath` already have the maximum amount of links pointing to them.

sys_enotdirpart of `Old` or `NewPath` is not directory.

sys_efault For the `pchar` case: One of the pointers points to an invalid address.

sys_eaccessaccess is denied when attempting to move the file.

sys_enametoolong Either `Old` or `NewPath` is too long.

sys_enoenta directory component in `Old` or `NewPath` didn't exist.

sys_enomemnot enough kernel memory.

sys_erofs`NewPath` or `Old` is on a read-only file system.

sys_elooptoo many symbolic links were encountered trying to expand `Old` or `NewPath`

sys_enospcthe file system has no room for the new directory entry.

See also: FpUnLink (234)

1.4.61 FpRmdir

Synopsis: Remove a directory.

Declaration:
```
function FpRmdir(path: PChar) : cint
function FpRmdir(const path: RawByteString) : cint
```

Visibility: default

Description: `FpRmdir` removes the directory `Path` from the system. The directory must be empty for this call to succeed, and the user must have the necessary permissions in the parent directory. Only the last component of the directory is removed, i.e. higher-lying directories are not removed.

On success, zero is returned. A nonzero return value indicates failure.

Note: There exist a portable alternative to fpRmDir: system.rmdir. Please use fpRmDir only if you are writing Unix specific code. System.rmdir will work on all operating systems.

Errors: Extended error information can be retrieved using fpGetErrno (198).

1.4.62 fpSelect

Synopsis: Wait for events on file descriptors.

Declaration:
```
function FPSelect(N: cint; readfds: pFDSet; writefds: pFDSet;
                  exceptfds: pFDSet; TimeOut: ptimeval) : cint
function fpSelect(N: cint; readfds: pFDSet; writefds: pFDSet;
                  exceptfds: pFDSet; TimeOut: cint) : cint
function fpSelect(var T: Text; TimeOut: ptimeval) : cint
function fpSelect(var T: Text; TimeOut: time_t) : cint
```

Visibility: default

Description: `FpSelect` checks one of the file descriptors in the `FDSets` to see if the following I/O operation on the file descriptors will block.

`readfds, writefds` and `exceptfds` are pointers to arrays of 256 bits. If you want a file descriptor to be checked, you set the corresponding element in the array to 1. The other elements in the array must be set to zero. Three arrays are passed : The entries in `readfds` are checked to see if the following read operation will block. The entries in `writefds` are checked to see if the following write operation will block, while entries in `exceptfds` are checked to see if an exception occurred on them.

You can use the functions fpFD_ZERO (195), fpFD_Clr (194), fpFD_Set (194) or fpFD_IsSet (194) to manipulate the individual elements of a set.

The pointers can be `Nil`.

N is the value of the largest file descriptor in one of the sets, + 1. In other words, it is the position of the last bit which is set in the array of bits.

`TimeOut` can be used to set a time limit. If `TimeOut` can be two types :

1. `TimeOut` is of type `ptimeval` and contains a zero time, the call returns immediately. If `TimeOut` is `Nil`, the kernel will wait forever, or until a status changed.
2. `TimeOut` is of type `cint`. If it is -1, this has the same effect as a `Timeout` of type `PTime` which is `Nil`. Otherwise, `TimeOut` contains a time in milliseconds.

When the TimeOut is reached, or one of the file descriptors has changed, the `Select` call returns. On return, it will have modified the entries in the array which have actually changed, and it returns the number of entries that have been changed. If the timeout was reached, and no descriptor changed, zero is returned; The arrays of indexes are undefined after that. On error, -1 is returned.

The variant with the text file will execute the `FpSelect` call on the file descriptor associated with the text file `T`

Errors: On error, the function returns -1. Extended error information can be retrieved using fpGetErrno (198).

SYS_EBADF An invalid descriptor was specified in one of the sets.

SYS_EINTR A non blocked signal was caught.

SYS_EINVAL `N` is negative or too big.

SYS_ENOMEM `Select` was unable to allocate memory for its internal tables.

See also: fpFD_ZERO (195), fpFD_Clr (194), fpFD_Set (194), fpFD_IsSet (194)

Listing: ./examples/bunixex/ex33.pp

```
Program Example33;

{ Program to demonstrate the Select function. }

Uses BaseUnix;

Var FDS : Tfdset;

begin
  fpfd_zero(FDS);
  fpfd_set(0,FDS);
  Writeln ('Press the <ENTER> to continue the program.');
  { Wait until File descriptor 0 (=Input) changes }
  fpSelect (1,@FDS,nil,nil,nil);
  { Get rid of <ENTER> in buffer }
  readln;
  Writeln ('Press <ENTER> key in less than 2 seconds...');
  Fpfd_zero(FDS);
  FpFd_set (0,FDS);
  if fpSelect (1,@FDS,nil,nil,2000)>0 then
    Writeln ('Thank you !')
    { FD_ISSET(0,FDS) would be true here. }
  else
    Writeln ('Too late !');
end.
```

1.4.63 fpseterrno

Synopsis: Set extended error information.

Declaration: `procedure fpseterrno(err: LongInt)`

Visibility: default

Description: `fpseterrno` sets the extended information on the latest error. It is called by all functions that communicate with the kernel or C library.

Unless a direct kernel call is performed, there should never be any need to call this function.

See also: fpgeterrno (198)

1.4.64 FpSetgid

Synopsis: Set the current group ID.

Declaration: `function FpSetgid(gid: TGid) : cint`

Visibility: default

Description: `fpSetUID` sets the group ID of the current process. This call will only work if it is executed as root, or the program is setgid root.

On success, zero is returned, on error -1 is returned.

Errors: Extended error information can be retrieved with fpGetErrNo (198).

See also: FpSetUid (222), FpGetGid (199), FpGetUid (202), FpGetEUid (198), FpGetEGid (197), FpGetPid (200), FpGetPPid (200)

1.4.65 fpSetPriority

Synopsis: Set process priority.

Declaration: `function fpSetPriority(Which: cint; Who: cint; What: cint) : cint`

Visibility: default

Description: `fpSetPriority` sets the priority with which a process is running. Which process(es) is determined by the `Which` and `Who` variables. `Which` can be one of the predefined constants:

Prio_Process`Who` is interpreted as process ID

Prio_PGrp`Who` is interpreted as process group ID

Prio_User`Who` is interpreted as user ID

`Prio` is a value in the range -20 to 20.

For an example, see FpNice (210).

The function returns zero on success, -1 on failure

Errors: Extended error information is returned by the FpGetErrno (198) function.

sys_esrch No process found using `which` and `who`.

sys_einval`Which` was not one of `Prio_Process`, `Prio_Grp` or `Prio_User`.

sys_epermA process was found, but neither its effective or real user ID match the effective user ID of the caller.

sys_eaccesA non-superuser tried to a priority increase.

See also: FpGetPriority (201), FpNice (210)

1.4.66 FpSetRLimit

Synopsis: Set process resource limits.

Declaration: `function FpSetRLimit(Resource: cint; rlim: PRLimit) : cint`

Visibility: default

Description: `FpGetRLimit` sets the resource limits for the current process: `resource` determines the resource of which the kernel should set the limits (one of the many `RLIMIT_*` constants). `rlim` should point to a TRLimit (243) record which contains the new limits for the resource indicated in `resource`.

The function returns zero if the resource limits were successfully set.

Errors: On error, -1 is returned and fpgeterrno (198) can be used to retrieve the error code.

See also: FpGetRLimit (201)

1.4.67 FpSetsid

Synopsis: Create a new session.

Declaration: `function FpSetsid : TPid`

Visibility: default

Description: `FpSetsid` creates a new session (process group). It returns the new process group id (as returned by FpGetpgrp (199). This call will fail if the current process is already the process group leader.

Errors: On error, -1 is returned. Extended error information can be retrieved with fpGetErrNo (198)

1.4.68 fpsettimeofday

Synopsis: Set kernel time.

Declaration: `function fpsettimeofday(tp: ptimeval; tzp: ptimezone) : cint`

Visibility: default

Description: `FpSetTimeOfDay` sets the kernel time to the number of seconds since 00:00, January 1 1970, GMT specified in the `tp` record. This time NOT corrected any way, not taking into account time-zones, daylight savings time and so on.

It is simply a wrapper to the kernel system call.

See also: #rtl.unix.FPGetTimeOfDay (2160)

1.4.69 FpSetuid

Synopsis: Set the current user ID.

Declaration: `function FpSetuid(uid: TUid) : cint`

Visibility: default

Description: `fpSetUID` sets the user ID of the current process. This call will only work if it is executed as root, or the program is setuid root.

On success, zero is returned, on error -1 is returned.

Errors: Extended error information can be retrieved with fpGetErrNo (198).

See also: FpGetGid (199), FpGetUid (202), FpGetEUid (198), FpGetEGid (197), FpGetPid (200), FpGetPPid (200), FpSetGid (221)

1.4.70 FPSigaction

Synopsis: Install signal handler.

Declaration:
```
function FPSigaction(sig: cint; act: psigactionrec; oact: psigactionrec)
                          : cint
```

Visibility: default

Description: `FPSigaction` changes the action to take upon receipt of a signal. `Act` and `Oact` are pointers to a SigActionRec (240) record. `Sig` specifies the signal, and can be any signal except **SIGKILL** or **SIGSTOP**.

If `Act` is non-nil, then the new action for signal `Sig` is taken from it. If `OAct` is non-nil, the old action is stored there. `Sa_Handler` may be `SIG_DFL` for the default action or `SIG_IGN` to ignore the signal. `Sa_Mask` Specifies which signals should be ignored during the execution of the signal handler. `Sa_Flags` Specifies a series of flags which modify the behaviour of the signal handler. You can 'or' none or more of the following :

SA_NOCLDSTOP If `sig` is **SIGCHLD** do not receive notification when child processes stop.

SA_ONESHOT or SA_RESETHAND Restore the signal action to the default state once the signal handler has been called.

SA_RESTART For compatibility with BSD signals.

SA_NOMASK or SA_NODEFER Do not prevent the signal from being received from within its own signal handler.

Errors: Extended error information can be retrieved using fpGetErrno (198).

sys_einvalan invalid signal was specified, or it was **SIGKILL** or **SIGSTOP**.

sys_efault`Act,OldAct` point outside this process address space

sys_eintr System call was interrupted.

See also: FpSigProcMask (226), FpSigPending (226), FpSigSuspend (227), FpKill (203)

Listing: ./examples/bunixex/ex57.pp

```
Program example57;

{ Program to demonstrate the SigAction function.}

{
do a kill -USR1 pid from another terminal to see what happens.
replace pid with the real pid of this program.
You can get this pid by running 'ps'.
}

uses BaseUnix;

Var
  oa,na : PSigActionRec;

Procedure DoSig(sig : cint);cdecl;

begin
  writeln('Receiving signal: ',sig);
end;
```

```
begin
  new(na);
  new(oa);
  na^.sa_Handler:=SigActionHandler(@DoSig);
  fillchar(na^.Sa_Mask,sizeof(na^.sa_mask),#0);
  na^.Sa_Flags:=0;
  {$ifdef Linux}                // Linux specific
    na^.Sa_Restorer:=Nil;
  {$endif}
  if fpSigAction(SigUsr1,na,oa)<>0 then
    begin
    writeln('Error: ',fpgeterrno,'.');
    halt(1);
    end;
  Writeln ('Send USR1 signal or press <ENTER> to exit');
  readln;
end.
```

1.4.71 FpSigAddSet

Synopsis: Set a signal in a signal set.

Declaration: `function FpSigAddSet(var nset: tsigset; signo: cint) : cint`

Visibility: default

Description: `FpSigAddSet` adds signal `Signo` to the signal set `nset`. The function returns 0 on success.

Errors: If an invalid signal number is given, -1 is returned.

See also: FpSigEmptySet (224), FpSigFillSet (225), FpSigDelSet (224), FpSigIsMember (225)

1.4.72 FpSigDelSet

Synopsis: Remove a signal from a signal set.

Declaration: `function FpSigDelSet(var nset: tsigset; signo: cint) : cint`

Visibility: default

Description: `FpSigDelSet` removes signal `Signo` to the signal set `nset`. The function returns 0 on success.

Errors: If an invalid signal number is given, -1 is returned.

See also: FpSigEmptySet (224), FpSigFillSet (225), FpSigAddSet (224), FpSigIsMember (225)

1.4.73 FpsigEmptySet

Synopsis: Clear all signals from signal set.

Declaration: `function FpsigEmptySet(var nset: tsigset) : cint`

Visibility: default

Description: `FpSigEmptySet` clears all signals from the signal set `nset`.

Errors: None. This function always returns zero.

See also: FpSigFillSet (225), FpSigAddSet (224), FpSigDelSet (224), FpSigIsMember (225)

1.4.74 FpSigFillSet

Synopsis: Set all signals in signal set.

Declaration: `function FpSigFillSet(var nset: tsigset) : cint`

Visibility: default

Description: `FpSigFillSet` sets all signals in the signal set `nset`.

Errors: None. This function always returns zero.

See also: FpSigEmptySet (224), FpSigAddSet (224), FpSigDelSet (224), FpSigIsMember (225)

1.4.75 FpSigIsMember

Synopsis: Check whether a signal appears in a signal set.

Declaration: `function FpSigIsMember(const nset: tsigset; signo: cint) : cint`

Visibility: default

Description: `FpSigIsMember` checks whether `SigNo` appears in the set `nset`. If it is a member, then 1 is returned. If not, zero is returned.

Errors: If an invalid signal number is given, -1 is returned.

See also: FpSigEmptySet (224), FpSigFillSet (225), FpSigAddSet (224), FpSigDelSet (224)

1.4.76 FpSignal

Synopsis: Install signal handler (deprecated).

Declaration:
```
function FpSignal(signum: LongInt; Handler: signalhandler)
                        : signalhandler
```

Visibility: default

Description: `FPSignal` installs a new signal handler (specified by `Handler`) for signal `SigNum`.

This call has a subset of the functionality provided by the FpSigAction (223) call. The return value for `FpSignal` is the old signal handler, or nil on error.

Errors: Extended error information can be retrieved using fpGetErrno (198).

SIG_ERR An error occurred.

See also: FpSigAction (223), FpKill (203)

Listing: ./examples/bunixex/ex58.pp

```
Program example58;

{ Program to demonstrate the Signal function.}

{
do a kill -USR1 pid from another terminal to see what happens.
replace pid with the real pid of this program.
You can get this pid by running 'ps'.
}
```

```
uses BaseUnix;

Procedure DoSig(sig : cint);cdecl;

begin
   writeln('Receiving signal: ',sig);
end;

begin
   if fpSignal(SigUsr1,SignalHandler(@DoSig))=signalhandler(SIG_ERR) then
     begin
     writeln('Error: ',fpGetErrno,'.');
     halt(1);
     end;
   Writeln ('Send USR1 signal or press <ENTER> to exit');
   readln;
end.
```

1.4.77 FpSigPending

Synopsis: Return set of currently pending signals.

Declaration: `function FpSigPending(var nset: tsigset) : cint`

Visibility: default

Description: `fpSigpending` allows the examination of pending signals (which have been raised while blocked.) The signal mask of pending signals is returned.

Errors: None

See also: fpSigAction (223), fpSigProcMask (226), fpSigSuspend (227), fpSignal (225), fpKill (203)

1.4.78 FpSigProcMask

Synopsis: Set list of blocked signals.

Declaration:
```
function FpSigProcMask(how: cint; nset: psigset; oset: psigset) : cint
function FpSigProcMask(how: cint; constref nset: tsigset;
                       var oset: tsigset) : cint
```

Visibility: default

Description: Changes the list of currently blocked signals. The behaviour of the call depends on `How` :

SIG_BLOCK The set of blocked signals is the union of the current set and the `nset` argument.

SIG_UNBLOCK The signals in `nset` are removed from the set of currently blocked signals.

SIG_SETMASK The list of blocked signals is set so `nset`.

If `oset` is non-nil, then the old set is stored in it.

Errors: `Errno` is used to report errors.

sys_efault `oset` or `nset` point to an address outside the range of the process.

sys_eintr System call was interrupted.

See also: fpSigAction (223), fpSigPending (226), fpSigSuspend (227), fpKill (203)

1.4.79 FpSigSuspend

Synopsis: Set signal mask and suspend process till signal is received.

Declaration:
```
function FpSigSuspend(const sigmask: tsigset) : cint
```

Visibility: default

Description: `fpSigSuspend` temporarily replaces the signal mask for the process with the one given in `SigMask`, and then suspends the process until a signal is received.

Errors: None

See also: fpSigAction (223), fpSigProcMask (226), fpSigPending (226), fpSignal (225), fpKill (203)

1.4.80 FpSigTimedWait

Synopsis: Wait for signal, with timeout.

Declaration:
```
function FpSigTimedWait(const sigset: tsigset; info: psiginfo;
                        timeout: ptimespec) : cint
```

Visibility: default

Description: `FpSigTimedWait` will suspend the current thread and wait for one of the signals in `sigset` to be delivered. information on the delivered signal is placed in the location provided by `info` (or in info itself, if the `Var` variant of the call is used). If the signal is not delivered within the time limit set in `timeout`, then the call will return -1, and `fpGetErrno` will return EAGAIN.

On success, the signal number is returned.

Errors: On error, -1 is returned, and extended error information can be obtained with `fpGetErrno`.

See also: FpSigSuspend (227)

1.4.81 FpSleep

Synopsis: Suspend process for several seconds.

Declaration:
```
function FpSleep(seconds: cuint) : cuint
```

Visibility: default

Description: `FpSleep` suspends the process till a time period as specified in `seconds` has passed, then the function returns. If the call was interrupted (e.g. by some signal) then the function may return earlier, and the return value is the remaining time till the end of the intended period.

If the function returns without error, the return value is zero.

See also: fpPause (213), fpAlarm (185), fpNanoSleep (209)

Listing: ./examples/bunixex/ex73.pp

```
program example73;

{ Program to demonstrate the FpSleep function. }

uses BaseUnix;

Var
```

```
  Res : Longint;

begin
  Write('Sleep returned : ');
  Flush(Output);
  Res:=(fpSleep(10));
  Writeln(res);
  If (res<>0) then
    Writeln('Remaining seconds      : ',res);
end.
```

1.4.82 FpStat

Synopsis: Retrieve file information about a file descriptor.

Declaration:
```
function FpStat(path: PChar; var buf: Stat) : cint
function FpStat(const path: RawByteString; var buf: Stat) : cint
function FpStat(path: ShortString; var buf: Stat) : cint
```

Visibility: default

Description: `FpFStat` gets information about the file specified in `Path`, and stores it in `Info`, which is of type stat (240). The function returns zero if the call was successful, a nonzero return value indicates failure.

Errors: Extended error information can be retrieved using fpGetErrno (198).

sys_enoent`Path` does not exist.

See also: FpStat (228), FpLStat (205)

Listing: ./examples/bunixex/ex28.pp

```
program example28;

{ Program to demonstrate the FStat function. }

uses BaseUnix;

var f : text;
    i : byte;
    info : stat;

begin
  { Make a file }
  assign (f,'test.fil');
  rewrite (f);
  for i:=1 to 10 do writeln (f,'Testline # ',i);
  close (f);
  { Do the call on made file. }
  if fpstat ('test.fil',info)<>0 then
     begin
       writeln('Fstat failed. Errno : ',fpgeterrno);
       halt (1);
     end;
  writeln;
  writeln ('Result of fstat on file ''test.fil''.');
```

```
  writeln ('Inode    : ',info.st_ino);
  writeln ('Mode     : ',info.st_mode);
  writeln ('nlink    : ',info.st_nlink);
  writeln ('uid      : ',info.st_uid);
  writeln ('gid      : ',info.st_gid);
  writeln ('rdev     : ',info.st_rdev);
  writeln ('Size     : ',info.st_size);
  writeln ('Blksize  : ',info.st_blksize);
  writeln ('Blocks   : ',info.st_blocks);
  writeln ('atime    : ',info.st_atime);
  writeln ('mtime    : ',info.st_mtime);
  writeln ('ctime    : ',info.st_ctime);
  { Remove file }
  erase (f);
end.
```

1.4.83 fpSymlink

Synopsis: Create a symbolic link.

Declaration: `function fpSymlink(oldname: PChar; newname: PChar) : cint`

Visibility: default

Description: `SymLink` makes `NewName` point to the file in `OldName`, which doesn't necessarily exist. The two files DO NOT have the same inode number. This is known as a 'soft' link.

The permissions of the link are irrelevant, as they are not used when following the link. Ownership of the file is only checked in case of removal or renaming of the link.

The function returns zero if the call was successful, a nonzero value if the call failed.

Errors: Extended error information is returned by the FpGetErrno (198) function.

sys_eperm The file system containing oldpath and newpath does not support linking files.

sys_eaccess Write access for the directory containing `Newpath` is disallowed, or one of the directories in `OldPath` or `NewPath` has no search (=execute) permission.

sys_enoent A directory entry in `OldPath` or `NewPath` does not exist or is a symbolic link pointing to a non-existent directory.

sys_enotdir A directory entry in `OldPath` or `NewPath` is nor a directory.

sys_enomem Insufficient kernel memory.

sys_erofs The files are on a read-only file system.

sys_eexist `NewPath` already exists.

sys_eloop `OldPath` or `NewPath` has a reference to a circular symbolic link, i.e. a symbolic link, whose expansion points to itself.

sys_enospc The device containing `NewPath` has no room for another entry.

See also: FpLink (203), FpUnLink (234), FpReadLink (216)

Listing: ./examples/unixex/ex22.pp

```
Program Example22;

{ Program to demonstrate the SymLink and UnLink functions. }
```

```
Uses baseunix,Unix;

Var F : Text;
    S : String;

begin
  Assign (F,'test.txt');
  Rewrite (F);
  Writeln (F,'This is written to test.txt');
  Close(f);
  { new.txt and test.txt are now the same file }
  if fpSymLink ('test.txt','new.txt')<>0 then
    writeln ('Error when symlinking !');
  { Removing test.txt still leaves new.txt
    Pointing now to a non-existent file ! }
  If fpUnlink ('test.txt')<>0 then
    Writeln ('Error when unlinking !');
  Assign (f,'new.txt');
  { This should fail, since the symbolic link
    points to a non-existent file! }
  {$i-}
  Reset (F);
  {$i+}
  If IOResult=0 then
    Writeln ('This shouldn''t happen');
 { Now remove new.txt also }
 If fpUnlink ('new.txt')<>0 then
   Writeln ('Error when unlinking !');
end.
```

1.4.84 fpS_ISBLK

Synopsis: Is file a block device.

Declaration: `function fpS_ISBLK(m: TMode) : Boolean`

Visibility: default

Description: `FpS_ISBLK` checks the file mode `m` to see whether the file is a block device file. If so it returns `True`.

See also: FpFStat (195), FpS_ISLNK (231), FpS_ISREG (232), FpS_ISDIR (231), FpS_ISCHR (230), FpS_ISFIFO (231), FpS_ISSOCK (232)

1.4.85 fpS_ISCHR

Synopsis: Is file a character device.

Declaration: `function fpS_ISCHR(m: TMode) : Boolean`

Visibility: default

Description: `FpS_ISCHR` checks the file mode `m` to see whether the file is a character device file. If so it returns `True`.

See also: FpFStat (195), FpS_ISLNK (231), FpS_ISREG (232), FpS_ISDIR (231), FpS_ISBLK (230), FpS_ISFIFO (231), FpS_ISSOCK (232)

1.4.86 fpS_ISDIR

Synopsis: Is file a directory.

Declaration: `function fpS_ISDIR(m: TMode) : Boolean`

Visibility: default

Description: `fpS_ISDIR` checks the file mode `m` to see whether the file is a directory. If so, it returns `True`

See also: FpFStat (195), FpS_ISLNK (231), FpS_ISREG (232), FpS_ISCHR (230), FpS_ISBLK (230), fpS_ISFIFO (231), FpS_ISSOCK (232)

1.4.87 fpS_ISFIFO

Synopsis: Is file a FIFO.

Declaration: `function fpS_ISFIFO(m: TMode) : Boolean`

Visibility: default

Description: `FpS_ISFIFO` checks the file mode `m` to see whether the file is a fifo (a named pipe). If so it returns `True`.

See also: FpFStat (195), FpS_ISLNK (231), FpS_ISREG (232), FpS_ISCHR (230), FpS_ISBLK (230), FpS_ISDIR (231), FpS_ISSOCK (232)

1.4.88 fpS_ISLNK

Synopsis: Is file a symbolic link.

Declaration: `function fpS_ISLNK(m: TMode) : Boolean`

Visibility: default

Description: `FpS_ISLNK` checks the file mode `m` to see whether the file is a symbolic link. If so it returns `True`

See also: FpFStat (195), FpS_ISFIFO (231), FpS_ISREG (232), FpS_ISCHR (230), FpS_ISBLK (230), FpS_ISDIR (231), FpS_ISSOCK (232)

Listing: ./examples/bunixex/ex53.pp

```
Program Example53;

{ Program to demonstrate the S_ISLNK function. }

Uses BaseUnix,Unix;

Var Info : Stat;

begin
  if fpLStat (paramstr(1),@info)=0 then
    begin
    if fpS_ISLNK(info.st_mode) then
      Writeln ('File is a link');
    if fpS_ISREG(info.st_mode) then
      Writeln ('File is a regular file');
    if fpS_ISDIR(info.st_mode) then
      Writeln ('File is a directory');
```

```
    if fpS_ISCHR(info.st_mode) then
      WriteIn ('File is a character device file');
    if fpS_ISBLK(info.st_mode) then
      WriteIn ('File is a block device file');
    if fpS_ISFIFO(info.st_mode) then
      WriteIn ('File is a named pipe (FIFO)');
    if fpS_ISSOCK(info.st_mode) then
      WriteIn ('File is a socket');
   end;
end.
```

1.4.89 fpS_ISREG

Synopsis: Is file a regular file.

Declaration: `function fpS_ISREG(m: TMode) : Boolean`

Visibility: default

Description: `FpS_ISREG` checks the file mode `m` to see whether the file is a regular file. If so it returns `True`

See also: FpFStat (195), FpS_ISFIFO (231), FpS_ISLNK (231), FpS_ISCHR (230), FpS_ISBLK (230), FpS_ISDIR (231), FPS_ISSOCK (232)

1.4.90 fpS_ISSOCK

Synopsis: Is file a Unix socket.

Declaration: `function fpS_ISSOCK(m: TMode) : Boolean`

Visibility: default

Description: `FpS_ISSOCK` checks the file mode `m` to see whether the file is a socket. If so it returns `True`.

See also: FpFStat (195), FpS_ISFIFO (231), FpS_ISLNK (231), FpS_ISCHR (230), FpS_ISBLK (230), FpS_ISDIR (231), FpS_ISREG (232)

1.4.91 fptime

Synopsis: Return the current Unix time.

Declaration:
```
function FpTime(var tloc: TTime) : TTime
function fptime : time_t
```

Visibility: default

Description: `FpTime` returns the number of seconds since 1970-01-01T00:00Z ignoring leap seconds. It is adjusted to the local time zone, but not to DST (daylight savings time). The result is also stored in `tloc` if it is specified.

Errors: On error, -1 is returned. Extended error information can be retrieved using fpGetErrno (198).

See also: #rtl.dateutils.unixtodatetime (680), #rtl.dateutils.unixtimestamptomac (680)

Listing: ./examples/bunixex/ex1.pp

```
Program Example1;

{ Program to demonstrate the fptime function. }

Uses baseunix;

begin
  Write ('Second past the start of the Epoch (1970-01-01T00:00): ');
  WriteIn (fptime);
end.
```

1.4.92 FpTimes

Synopsis: Return execution times for the current process.

Declaration: `function FpTimes(var buffer: tms) : TClock`

Visibility: default

Description: `fpTimes` stores the execution time of the current process and child processes in `buffer`.

The return value (on Linux) is the number of clock ticks since boot time. On error, -1 is returned, and extended error information can be retrieved with fpGetErrno (198).

See also: fpUTime (234)

1.4.93 FpUmask

Synopsis: Set file creation mask.

Declaration: `function FpUmask(cmask: TMode) : TMode`

Visibility: default

Description: `fpUmask` changes the file creation mask for the current user to `cmask`. The current mask is returned.

See also: fpChmod (186)

Listing: ./examples/bunixex/ex27.pp

```
Program Example27;

{ Program to demonstrate the Umask function. }

Uses BaseUnix;

begin
  WriteIn ('Old Umask was : ',fpUmask(&111));
  WRiteIn ('New Umask is  : ',&111);
end.
```

1.4.94 FpUname

Synopsis: Return system name.

Declaration:
```
function FpUname(var name: UtsName) : cint
```

Visibility: default

Description: `Uname` gets the name and configuration of the current Linux kernel, and returns it in the `name` record.

On success, 0 is returned, on error, -1 is returned.

Errors: Extended error information can be retrieved using fpGetErrno (198).

See also: FpUTime (234)

1.4.95 FpUnlink

Synopsis: Unlink (i.e. remove) a file.

Declaration:
```
function FpUnlink(path: PChar) : cint
function FpUnlink(const path: RawByteString) : cint
```

Visibility: default

Description: `FpUnlink` decreases the link count on file `Path`. `Path` can be of type `AnsiString` or `PChar`. If the link count is zero, the file is removed from the disk.

The function returns zero if the call was successful, a nonzero value indicates failure.

Note: There exist a portable alternative to erase files: system.erase. Please use fpUnlink only if you are writing Unix specific code. System.erase will work on all operating systems.

For an example, see FpLink (203).

Errors: Extended error information can be retrieved using fpGetErrno (198).

sys_eaccess You have no write access right in the directory containing `Path`, or you have no search permission in one of the directory components of `Path`.

sys_eperm The directory containing pathname has the sticky-bit set and the process's effective uid is neither the uid of the file to be deleted nor that of the directory containing it.

sys_enoent A component of the path doesn't exist.

sys_enotdir A directory component of the path is not a directory.

sys_eisdir `Path` refers to a directory.

sys_enomem Insufficient kernel memory.

sys_erofs `Path` is on a read-only file system.

See also: FpLink (203), FpSymLink (229)

1.4.96 FpUtime

Synopsis: Set access and modification times of a file (touch).

Declaration:
```
function FpUtime(path: PChar; times: pUtimBuf) : cint
function FpUtime(const path: RawByteString; times: pUtimBuf) : cint
```

Visibility: default

Description: `FpUtime` sets the access and modification times of the file specified in `Path`. the `times` record contains 2 fields, `actime`, and `modtime`, both of type `time_t` (commonly a longint). They should be filled with an epoch-like time, specifying, respectively, the last access time, and the last modification time. For some file systems (most notably, FAT), these times are the same.

The function returns zero on success, a nonzero return value indicates failure.

Errors: Extended error information can be retrieved using fpGetErrno (198).

sys_eaccess One of the directories in `Path` has no search (=execute) permission.

sys_enoentA directory entry in `Path` does not exist or is a symbolic link pointing to a non-existent directory.

Other errors may occur, but aren't documented.

See also: FpTime (232), FpChown (187), FpAccess (184)

Listing: ./examples/bunixex/ex25.pp

```
Program Example25;

{ Program to demonstrate the UTime function. }

Uses Dos,BaseUnix,Unix,UnixUtil;

Var utim : utimbuf;
    dow,msec,year,month,day,hour,minute,second : Word;

begin
  { Set access and modification time of executable source }
  GetTime (hour,minute,second,msec);
  GetDate (year,month,day,dow);
  utim.actime:=LocalToEpoch(year,month,day,hour,minute,second);
  utim.modtime:=utim.actime;
  if Fputime('ex25.pp',@utim)<>0 then
    writeln ('Call to UTime failed !')
  else
    begin
    Write ('Set access and modification times to : ');
    Write (Hour:2,':',minute:2,':',second,', ');
    WriteIn (Day:2,'/',month:2,'/',year:4);
    end;
end.
```

1.4.97 FpWait

Synopsis: Wait for a child to exit.

Declaration: `function FpWait(var stat_loc: cint) : TPid`

Visibility: default

Description: `fpWait` suspends the current process and waits for any child to exit or stop due to a signal. It reports the exit status of the exited child in `stat_loc`.

The return value of the function is the process ID of the child that exited, or -1 on error.

Errors: Extended error information can be retrieved using fpgetErrno (198).

See also: fpFork (195), fpExecve (192), fpWaitPid (236)

1.4.98 FpWaitPid

Synopsis: Wait for a process to terminate.

Declaration:
```
function FpWaitpid(pid: TPid; stat_loc: pcint; options: cint) : TPid
function FpWaitPid(pid: TPid; var Status: cint; Options: cint) : TPid
```

Visibility: default

Description: `fpWaitPid` waits for a child process with process ID `Pid` to exit. The value of `Pid` can be one of the following:

Pid < -1 Causes `fpWaitPid` to wait for any child process whose process group ID equals the absolute value of `pid`.

Pid = -1 Causes `fpWaitPid` to wait for any child process.

Pid = 0 Causes `fpWaitPid` to wait for any child process whose process group ID equals the one of the calling process.

Pid > 0 Causes `fpWaitPid` to wait for the child whose process ID equals the value of `Pid`.

The `Options` parameter can be used to specify further how `fpWaitPid` behaves:

WNOHANG Causes `fpWaitpid` to return immediately if no child hasexited.

WUNTRACED Causes `fpWaitPid` to return also for children which are stopped, but whose status has not yet been reported.

__WCLONE Causes `fpWaitPid` also to wait for threads created by the #rtl.linux.Clone (994) call.

The exit status of the process that caused `fpWaitPID` is reported in `stat_loc` or `Status`.

Upon return, it returns the process id of the process that exited, 0 if no process exited, or -1 in case of failure.

For an example, see fpFork (195).

Errors: Extended error information can be retrieved using fpgetErrno (198).

See also: fpFork (195), fpExecve (192), fpWait (235)

1.4.99 FpWrite

Synopsis: Write data to file descriptor.

Declaration:
```
function FpWrite(fd: cint; buf: PChar; nbytes: TSize) : TsSize
function FpWrite(fd: cint; const buf; nbytes: TSize) : TsSize
```

Visibility: default

Description: `FpWrite` writes at most `nbytes` bytes from `buf` to file descriptor `fd`.

The function returns the number of bytes actually written, or -1 if an error occurred.

Errors: Extended error information can be retrieved using fpGetErrno (198).

See also: FpOpen (211), FpClose (188), FpRead (215), FpFTruncate (196), FpLSeek (204)

1.4.100 FpWriteV

Synopsis: Vector write: Write from multiple buffers to a file descriptor.

Declaration: `function FpWriteV(fd: cint; const iov: piovec; iovcnt: cint) : TsSize`

Visibility: default

Description: `FpWriteV` writes data to file descriptor `fd`. The data is taken from `iovcnt` buffers described by the tiovec (182) buffers pointed to by `iov`. It works like fpWrite (236) only from multiple buffers.

Errors: On error, -1 is returned.

See also: FpReadV (218), FpPWrite (215), FpPRead (214)

1.4.101 FreeShellArgV

Synopsis: Free the result of a CreateShellArgV (184) function.

Declaration: `procedure FreeShellArgV(p: PPChar)`

Visibility: default

Description: `FreeShellArgV` frees the memory pointed to by `P`, which was allocated by a call to CreateShellArgV (184).

Errors: None.

See also: CreateShellArgV (184)

1.4.102 wexitStatus

Synopsis: Extract the exit status from the fpWaitPID (236) result.

Declaration: `function wexitStatus(Status: cint) : cint`

Visibility: default

Description: `WEXITSTATUS` can be used to extract the exit status from `Status`, the result of the FpWaitPID (236) call.

See also: FpWaitPID (236), WTERMSIG (238), WSTOPSIG (238), WIFEXITED (237), WIFSIGNALED (238)

1.4.103 wifexited

Synopsis: Check whether the process exited normally.

Declaration: `function wifexited(Status: cint) : Boolean`

Visibility: default

Description: `WIFEXITED` checks `Status` and returns `True` if the status indicates that the process terminated normally, i.e. was not stopped by a signal.

See also: FpWaitPID (236), WTERMSIG (238), WSTOPSIG (238), WIFSIGNALED (238), WEXITSTATUS (237)

1.4.104 wifsignaled

Synopsis: Check whether the process was exited by a signal.

Declaration: `function wifsignaled(Status: cint) : Boolean`

Visibility: default

Description: `WIFSIGNALED` returns `True` if `Status` indicates that the process exited because it received a signal.

See also: FpWaitPID (236), WTERMSIG (238), WSTOPSIG (238), WIFEXITED (237), WEXITSTATUS (237)

1.4.105 wstopsig

Synopsis: Return the exit code from the process.

Declaration: `function wstopsig(Status: cint) : cint`

Visibility: default

Description: `WSTOPSIG` is an alias for WEXITSTATUS (237).

See also: FpWaitPID (236), WTERMSIG (238), WIFEXITED (237), WIFSIGNALED (238), WEXITSTATUS (237)

1.4.106 wtermsig

Synopsis: Return the signal that caused a process to exit.

Declaration: `function wtermsig(Status: cint) : cint`

Visibility: default

Description: `WTERMSIG` extracts from `Status` the signal number which caused the process to exit.

See also: FpWaitPID (236), WSTOPSIG (238), WIFEXITED (237), WIFSIGNALED (238), WEXITSTATUS (237)

1.5 Dir

```
Dir = record
  dd_fd : LongInt;
  dd_loc : LongInt;
  dd_size : LongInt
  ;
  dd_buf : pDirent;
  dd_nextoff : Cardinal;
  dd_max : Integer
  ;
  dd_lock : pointer;
end
```

Record used in fpOpenDir (212) and fpReadDir (216) calls.

1.6 Dirent

```
Dirent = record
  d_fileno : ino64_t;
  d_off : off_t;
  d_reclen
  : cushort;
  d_type : cuchar;
  d_name : Array[0..4095-sizeof(ino64_t
  )-sizeof(off_t)-sizeof(cushort)-sizeof(cuchar)] of Char;
end
```

Record used in the fpReadDir (216) function to return files in a directory.

1.7 FLock

```
FLock = record
  l_type : cshort;
  l_whence : cshort;
  l_start
  : kernel_off_t;
  l_len : kernel_off_t;
  l_pid : pid_t;
end
```

Lock description type for fpFCntl (193) lock call.

1.8 iovec

```
iovec = record
  iov_base : pointer;
  iov_len : size_t;
end
```

`iovec` is used in fpreadv (218) for IO to multiple buffers to describe a buffer location.

1.9 pollfd

```
pollfd = record
  fd : cint;
  events : cshort;
  revents : cshort
  ;
end
```

`pollfd` is used in the fpPoll (214) call to describe the various actions.

1.10 sigactionrec

```
sigactionrec = record
  sa_handler : sigactionhandler_t;
  sa_flags
  : culong;
  sa_restorer : sigrestorerhandler_t;
  sa_mask : sigset_t
  ;
end
```

Record used in fpSigAction (223) call.

1.11 Stat

```
Stat = packed record
  st_dev : QWord;
  __pad0_ : Array[0..3] of
  Byte;
  __st_ino_ : Cardinal;
  st_mode : Cardinal;
  st_nlink :
  Cardinal;
  st_uid : Cardinal;
  st_gid : Cardinal;
  st_rdev : QWord
  ;
  __pad3_ : Array[0..3] of Byte;
  st_size : QWord;
  st_blksize
  : Cardinal;
  st_blocks : QWord;
  st_atime : Cardinal;
  st_atime_nsec
  : Cardinal;
  st_mtime : Cardinal;
  st_mtime_nsec : Cardinal;
  st_ctime : Cardinal;
  st_ctime_nsec : Cardinal;
  st_ino : QWord
  ;
end
```

Record describing an inode (file) in the FPFstat (195) call.

1.12 tfpreg

```
tfpreg = record
  significand : Array[0..3] of Word;
  exponent :
  Word;
```

```
end
```

Record describing floating point register in signal handler.

1.13 tfpstate

```
tfpstate = record
  cw : Cardinal;
  sw : Cardinal;
  tag : Cardinal
  ;
  ipoff : Cardinal;
  cssel : Cardinal;
  dataoff : Cardinal;
  datasel : Cardinal;
  st : Array[0..7] of tfpreg;
  status : Word
  ;
  magic : Word;
  fxsr_env : Array[0..5] of DWord;
  mxcsr : DWord
  ;
  reserved : DWord;
  fxsr_st : Array[0..7] of tfpxreg;
  xmmreg
  : Array[0..7] of txmmreg;
case Byte of
1: (
  padding : Array[0.
  .43] of DWord;
case Byte of
1: (
  padding2 : Array[0..11] of DWord
  ;
);
2: (
  sw_reserved : tfpx_sw_bytes;
);
);
2: (
  padding1 : Array
  [0..43] of DWord;
);
end
```

Record describing floating point unit in signal handler.

1.14 tfpxreg

```
tfpxreg = record
  significand : Array[0..3] of Word;
```

```
  exponent
  : Word;
  padding : Array[0..2] of Word;
end
```

Record describing 16-byte floating point register in signal handler.

1.15 tfpx_sw_bytes

```
tfpx_sw_bytes = record
  magic1 : DWord;
  extended_size : DWord;
  xfeatures : QWord;
  xstate_size : DWord;
  padding : Array[0.
  .6] of DWord;
end
```

Extended state information in signal handler.

1.16 timezone

```
timezone = record
  tz_minuteswest : cint;
  tz_dsttime : cint;
end
```

Record describing a timezone.

1.17 tms

```
tms = record
  tms_utime : clock_t;
  tms_stime : clock_t;
  tms_cutime
  : clock_t;
  tms_cstime : clock_t;
end
```

Record containing timings for fpTimes (233) call.

1.18 TRLimit

```
TRLimit = record
  rlim_cur : rlim_t;
  rlim_max : rlim_t;
end
```

TRLimit is the structure used by the kernel to return resource limit information in.

1.19 tsigaltstack

```
tsigaltstack = record
  ss_sp : pointer;
  ss_flags : LongInt;
  ss_size
  : LongInt;
end
```

Provide the location of an alternate signal handler stack.

1.20 TSigContext

```
TSigContext = record
  gs : Word;
  __gsh : Word;
  fs : Word;
  __fsh
  : Word;
  es : Word;
  __esh : Word;
  ds : Word;
  __dsh : Word
  ;
  edi : Cardinal;
  esi : Cardinal;
  ebp : Cardinal;
  esp : Cardinal
  ;
  ebx : Cardinal;
  edx : Cardinal;
  ecx : Cardinal;
  eax : Cardinal
  ;
  trapno : Cardinal;
  err : Cardinal;
  eip : Cardinal;
  cs :
  Word;
  __csh : Word;
  eflags : Cardinal;
  esp_at_signal : Cardinal
  ;
  ss : Word;
  __ssh : Word;
  fpstate : pfpstate;
  oldmask : Cardinal
  ;
  cr2 : Cardinal;
```

```
end
```

This type is CPU dependent. Cross-platform code should not use the contents of this record.

1.21 tsiginfo

```
tsiginfo = record
  si_signo : LongInt;
  si_errno : LongInt;
  si_code
  : LongInt;
  _sifields : record
  case LongInt of
  0: (
    _pad
  : Array[0..(SI_PAD_SIZE)-1] of LongInt;
  );
  1: (
    _kill : record
  _pid : pid_t;
      _uid : uid_t;
    end;
  );
  2: (
  _timer : record
      _timer1 : DWord;
      _timer2 : DWord;
  end;
  );
  3: (
    _rt : record
      _pid : pid_t;
      _uid
  : uid_t;
      _sigval : pointer;
    end;
  );
  4: (
    _sigchld
  : record
      _pid : pid_t;
      _uid : uid_t;
      _status :
  LongInt;
      _utime : clock_t;
      _stime : clock_t;
    end
  ;
  );
  5: (
    _sigfault : record
      _addr : pointer;
```

```
    end
  ;
  );
  6: (
    _sigpoll : record
      _band : LongInt;
      _fd
  : LongInt;
    end;
  );
  end;
end
```

This type describes the signal that occurred.

1.22 TUcontext

```
TUcontext = record
  uc_flags : Cardinal;
  uc_link : Pucontext;
  uc_stack : tsigaltstack;
  uc_mcontext : TSigContext;
  uc_sigmask
  : tsigset;
end
```

This structure is used to describe the user context in a program or thread. It is not used in this unit, but is provided for completeness.

1.23 txmmreg

```
txmmreg = record
  element : Array[0..3] of DWord;
end
```

Record describing 16-byte MMX register in signal handler.

1.24 UTimBuf

```
UTimBuf = record
  actime : time_t;
  modtime : time_t;
end
```

Record used in fpUtime (234) to set file access and modification times.

1.25 UtsName

```
UtsName = record
  Sysname : Array[0..UTSNAME_LENGTH-1] of Char;
  Nodename : Array[0..UTSNAME_NODENAME_LENGTH-1] of Char;
  Release
  : Array[0..UTSNAME_LENGTH-1] of Char;
  Version : Array[0..UTSNAME_LENGTH
  -1] of Char;
  Machine : Array[0..UTSNAME_LENGTH-1] of Char;
  Domain
  : Array[0..UTSNAME_DOMAIN_LENGTH-1] of Charplatform;
end
```

The elements of this record are null-terminated C style strings, you cannot access them directly. Note that the `Domain` field is a GNU extension, and may not be available on all platforms.

Chapter 2

Reference for unit 'Character'

2.1 Used units

Table 2.1: Used units by unit 'Character'

Name	Page
System	1362
unicodedata	2098

2.2 Overview

The character unit contains the TCharacter (255) class, which consists mainly of class functions. It should not be constructed, but its class methods can be used. All class methods also exist as regular methods.

Many routines depend on Unicode collation data to be present in the binary (or distributed on disc with the application. This data can be loaded using the routines in the unicodedata (2098) unit.. The FPC project distributes some Unicode collation data in `.bco` files which can be loaded using the LoadCollation (2109) routine from that unit.

2.3 Constants, types and variables

2.3.1 Types

```
TCharacterOption = (coIgnoreInvalidSequence)
```

Table 2.2: Enumeration values for type TCharacterOption

Value	Explanation
coIgnoreInvalidSequence	Ignore invalid unicodecode sequences.

`TCharacterOption` is used in the toUpper (255) and toLower (254) functions to control the behaviour of the function.

```
TCharacterOptions = Set of TCharacterOption
```

`TCharacterOptions` is the set of `TCharacterOption`, used in toUpper (255) and toLower (254) functions to control the behaviour of the function.

```
TUnicodeCategory = (ucUppercaseLetter,ucLowercaseLetter,
  ucTitlecaseLetter,ucModifierLetter,ucOtherLetter,
  ucNonSpacingMark,ucCombiningMark,ucEnclosingMark
  ,
                    ucDecimalNumber,ucLetterNumber,ucOtherNumber
  ,
                    ucConnectPunctuation,ucDashPunctuation,
  ucOpenPunctuation,ucClosePunctuation,
  ucInitialPunctuation,ucFinalPunctuation,
  ucOtherPunctuation,ucMathSymbol,ucCurrencySymbol,
  ucModifierSymbol,ucOtherSymbol,ucSpaceSeparator,
  ucLineSeparator,ucParagraphSeparator,ucControl,
  ucFormat,ucSurrogate,ucPrivateUse,ucUnassigned)
```

Table 2.3: Enumeration values for type TUnicodeCategory

Value	Explanation
ucClosePunctuation	Punctuation, close (Pe).
ucCombiningMark	Mark, spacing combining (Mc).
ucConnectPunctuation	Punctuation, connector (Pc).
ucControl	Other, control (Cc).
ucCurrencySymbol	Symbol, currency (Sc).
ucDashPunctuation	Punctuation, dash (Pd).
ucDecimalNumber	Number, decimal digit (Nd).
ucEnclosingMark	Mark, enclosing (Me).
ucFinalPunctuation	Punctuation, final quote (Pf, may behave like Ps or Pe depending on usage).
ucFormat	Other, format (Cf).
ucInitialPunctuation	Punctuation, initial quote (Pi, may behave like Ps or Pe depending on usage).
ucLetterNumber	Number, letter (Nl).
ucLineSeparator	Separator, line (Zl).
ucLowercaseLetter	Letter, lowercase (Ll).
ucMathSymbol	Symbol, math (Sm).
ucModifierLetter	Letter, modifier (Lm).
ucModifierSymbol	Symbol, modifier (Sk).
ucNonSpacingMark	Mark, nonspacing (Mn).
ucOpenPunctuation	Punctuation, open (Ps).
ucOtherLetter	Letter, other (Lo).
ucOtherNumber	Number, other (No).
ucOtherPunctuation	Punctuation, other (Po).
ucOtherSymbol	Symbol, other (So).
ucParagraphSeparator	Separator, paragraph (Zp).
ucPrivateUse	Other, private use (Co).
ucSpaceSeparator	Separator, space (Zs).
ucSurrogate	Other, surrogate (Cs).
ucTitlecaseLetter	Letter, titlecase (Lt).
ucUnassigned	Other, not assigned (including noncharacters) (Cn).
ucUppercaseLetter	Letter, uppercase (Lu).

This enumeration type contains the characterization of all possible Unicode characters. It is used in the GetUnicodeCategory (250) and TCharacter.GetUnicodeCategory (257) functions.

```
TUnicodeCategorySet = Set of TUnicodeCategory
```

`TUnicodeCategorySet` is the set of TUnicodeCategory (248). It is used internally in the TCharacter (255) class.

2.4 Procedures and functions

2.4.1 ConvertFromUtf32

Synopsis: alias for `TCharacter.ConvertFromUtf32`.

Declaration: `function ConvertFromUtf32(AChar: UCS4Char) : UnicodeString`

Visibility: default

Description: `ConvertFromUtf32` is a shortcut for TCharacter.ConvertFromUtf32 (256).

See also: TCharacter.ConvertFromUtf32 (256)

2.4.2 ConvertToUtf32

Synopsis: alias for `TCharacter.ConvertToUtf32`.

Declaration:
```
function ConvertToUtf32(const AString: UnicodeString; AIndex: Integer)
                         : UCS4Char;  Overload
function ConvertToUtf32(const AString: UnicodeString; AIndex: Integer;
                        out ACharLength: Integer) : UCS4Char;  Overload
function ConvertToUtf32(const AHighSurrogate: UnicodeChar;
                        const ALowSurrogate: UnicodeChar) : UCS4Char
                        ;  Overload
```

Visibility: default

Description: `ConvertToUtf32` is a shortcut for TCharacter.ConvertToUtf32 (256).

See also: TCharacter.ConvertToUtf32 (256)

2.4.3 GetNumericValue

Synopsis: Alias for `TCharacter.GetNumericValue`.

Declaration:
```
function GetNumericValue(AChar: UnicodeChar) : Double;  Overload
function GetNumericValue(const AString: UnicodeString; AIndex: Integer)
                         : Double;  Overload
```

Visibility: default

Description: `GetNumericValue` is a shortcut for TCharacter.GetNumericValue (257).

See also: TCharacter.GetNumericValue (257)

2.4.4 GetUnicodeCategory

Synopsis: Alias for `TCharacter.GetUnicodeCategory`.

Declaration:
```
function GetUnicodeCategory(AChar: UnicodeChar) : TUnicodeCategory
                            ;  Overload
function GetUnicodeCategory(const AString: UnicodeString;
                            AIndex: Integer) : TUnicodeCategory
                            ;  Overload
```

Visibility: default

Description: `GetUnicodeCategory` is a shortcut for TCharacter.GetUnicodeCategory (257).

See also: TCharacter.GetUnicodeCategory (257)

2.4.5 IsControl

Synopsis: Alias for `TCharacter.IsControl`.

Declaration:
```
function IsControl(AChar: UnicodeChar) : Boolean;  Overload
function IsControl(const AString: UnicodeString; AIndex: Integer)
                   : Boolean;  Overload
```

Visibility: default

Description: `IsControl` is a shortcut for TCharacter.IsControl (257).

See also: TCharacter.IsControl (257)

2.4.6 IsDigit

Synopsis: Alias for `TCharacter.IsDigit`.

Declaration:
```
function IsDigit(AChar: UnicodeChar) : Boolean;  Overload
function IsDigit(const AString: UnicodeString; AIndex: Integer)
                 : Boolean;  Overload
```

Visibility: default

Description: `IsDigit` is a shortcut for TCharacter.IsDigit (258).

See also: TCharacter.IsDigit (258)

2.4.7 IsHighSurrogate

Synopsis: Alias for `TCharacter.IsHighSurrogate`.

Declaration:
```
function IsHighSurrogate(AChar: UnicodeChar) : Boolean;  Overload
function IsHighSurrogate(const AString: UnicodeString; AIndex: Integer)
                         : Boolean;  Overload
```

Visibility: default

Description: `IsHighSurrogate` is a shortcut for TCharacter.IsHighSurrogate (259)

See also: TCharacter.IsHighSurrogate (259)

2.4.8 IsLetter

Synopsis: Alias for `TCharacter.IsLetter`.

Declaration:
```
function IsLetter(AChar: UnicodeChar) : Boolean;  Overload
function IsLetter(const AString: UnicodeString; AIndex: Integer)
                  : Boolean;  Overload
```

Visibility: default

Description: `IsLetter` is a shortcut for TCharacter.IsLetter (260)

See also: TCharacter.IsLetter (260)

2.4.9 IsLetterOrDigit

Synopsis: Alias for `TCharacter.IsLetterOrDigit`.

Declaration:
```
function IsLetterOrDigit(AChar: UnicodeChar) : Boolean;  Overload
function IsLetterOrDigit(const AString: UnicodeString; AIndex: Integer)
                         : Boolean;  Overload
```

Visibility: default

Description: `IsLetterOrDigit` is a shortcut for TCharacter.IsLetterOrDigit (260).

See also: TCharacter.IsLetterOrDigit (260)

2.4.10 IsLower

Synopsis: Alias for `TCharacter.IsLower`.

Declaration:
```
function IsLower(AChar: UnicodeChar) : Boolean;  Overload
function IsLower(const AString: UnicodeString; AIndex: Integer)
                 : Boolean;  Overload
```

Visibility: default

Description: `IsLower` is a shortcut for TCharacter.IsLower (261)

See also: TCharacter.IsLower (261)

2.4.11 IsLowSurrogate

Synopsis: Alias for `TCharacter.IsLowSurrogate`.

Declaration:
```
function IsLowSurrogate(AChar: UnicodeChar) : Boolean;  Overload
function IsLowSurrogate(const AString: UnicodeString; AIndex: Integer)
                        : Boolean;  Overload
```

Visibility: default

Description: `IsLowSurrogate` is a shortcut for TCharacter.IsLowSurrogate (259)

See also: TCharacter.IsLowSurrogate (259)

2.4.12 IsNumber

Synopsis: Alias for `TCharacter.IsNumber`.

Declaration:
```
function IsNumber(AChar: UnicodeChar) : Boolean;  Overload
function IsNumber(const AString: UnicodeString; AIndex: Integer)
                  : Boolean;  Overload
```

Visibility: default

Description: `IsNumber` is a shortcut for TCharacter.IsNumber (261)

See also: TCharacter.IsNumber (261)

2.4.13 IsPunctuation

Synopsis: Alias for `TCharacter.IsPunctuation`.

Declaration:
```
function IsPunctuation(AChar: UnicodeChar) : Boolean;  Overload
function IsPunctuation(const AString: UnicodeString; AIndex: Integer)
                       : Boolean;  Overload
```

Visibility: default

Description: `IsPunctuation` is a shortcut for TCharacter.IsPunctuation (262)

See also: TCharacter.IsPunctuation (262)

2.4.14 IsSeparator

Synopsis: Alias for `TCharacter.IsSeparator`.

Declaration:
```
function IsSeparator(AChar: UnicodeChar) : Boolean;  Overload
function IsSeparator(const AString: UnicodeString; AIndex: Integer)
                     : Boolean;  Overload
```

Visibility: default

Description: `IsSeparator` is a shortcut for TCharacter.IsSeparator (262)

See also: TCharacter.IsSeparator (262)

2.4.15 IsSurrogate

Synopsis: Alias for `TCharacter.IsSurrogate`.

Declaration:
```
function IsSurrogate(AChar: UnicodeChar) : Boolean;  Overload
function IsSurrogate(const AString: UnicodeString; AIndex: Integer)
                     : Boolean;  Overload
```

Visibility: default

Description: `IsSurrogate` is a shortcut for TCharacter.IsSurrogate (258).

See also: TCharacter.IsSurrogate (258)

2.4.16 IsSurrogatePair

Synopsis: Alias for `TCharacter.IsSurrogatePair`.

Declaration:
```
function IsSurrogatePair(const AHighSurrogate: UnicodeChar;
                         const ALowSurrogate: UnicodeChar) : Boolean
                         ;  Overload
function IsSurrogatePair(const AString: UnicodeString; AIndex: Integer)
                         : Boolean;  Overload
```

Visibility: default

Description: `IsSurrogatePair` is a shortcut for TCharacter.IsSurrogatePair (260)

See also: TCharacter.IsSurrogatePair (260)

2.4.17 IsSymbol

Synopsis: Alias for `TCharacter.IsSymbol`.

Declaration:
```
function IsSymbol(AChar: UnicodeChar) : Boolean;  Overload
function IsSymbol(const AString: UnicodeString; AIndex: Integer)
                  : Boolean;  Overload
```

Visibility: default

Description: `IsSymbol` is a shortcut for TCharacter.IsSymbol (263)

See also: TCharacter.IsSymbol (263)

2.4.18 IsUpper

Synopsis: Alias for `TCharacter.IsUpper`.

Declaration:
```
function IsUpper(AChar: UnicodeChar) : Boolean;  Overload
function IsUpper(const AString: UnicodeString; AIndex: Integer)
                 : Boolean;  Overload
```

Visibility: default

Description: `IsUpper` is a shortcut for TCharacter.IsUpper (263)

See also: TCharacter.IsUpper (263)

2.4.19 IsWhiteSpace

Synopsis: Alias for `TCharacter.IsWhiteSpace`.

Declaration:
```
function IsWhiteSpace(AChar: UnicodeChar) : Boolean;  Overload
function IsWhiteSpace(const AString: UnicodeString; AIndex: Integer)
                      : Boolean;  Overload
```

Visibility: default

Description: `IsWhiteSpace` is a shortcut for TCharacter.IsWhiteSpace (263)

See also: TCharacter.IsWhiteSpace (263)

2.4.20 ToLower

Synopsis: Alias for `TCharacter.ToLower`.

Declaration:
```
function ToLower(AChar: UnicodeChar) : UnicodeChar;  Overload
function ToLower(const AString: UnicodeString) : UnicodeString
                ;  Overload
```

Visibility: default

Description: `ToLower` is a shortcut for TCharacter.ToLower (264)

See also: TCharacter.ToLower (264)

2.4.21 ToUpper

Synopsis: Alias for `TCharacter.ToUpper`.

Declaration:
```
function ToUpper(AChar: UnicodeChar) : UnicodeChar;  Overload
function ToUpper(const AString: UnicodeString) : UnicodeString
                ;  Overload
```

Visibility: default

Description: `ToUpper` is a shortcut for TCharacter.ToUpper (264)

See also: TCharacter.ToUpper (264)

2.5 TCharacter

2.5.1 Description

`TCharacter` is provided for Delphi compatibility. All it's class functions and methods are also available as regular functions.

2.5.2 Method overview

Page	Method	Description
256	ConvertFromUtf32	Convert a UTF32 character to UnicodeString.
256	ConvertToUtf32	Convert a UTF16 character to a UTF32 character.
255	Create	Constructor (do not call).
257	GetNumericValue	Get the numeric value of the character.
257	GetUnicodeCategory	Get the Unicode category of a character.
257	IsControl	Check whether a Unicode character is a Unicode control character.
258	IsDigit	Check whether a Unicode character is a digit.
259	IsHighSurrogate	Check whether a Unicode character is a surrogate in the high range.
260	IsLetter	Check if a Unicode character is a letter.
260	IsLetterOrDigit	Check if a Unicode character is a letter or digit.
261	IsLower	Check if a Unicode character is a lowercase letter.
259	IsLowSurrogate	Check whether a Unicode character is a surrogate in the low range.
261	IsNumber	Check if a Unicode character is a number.
262	IsPunctuation	Check if a Unicode character is a punctuation character.
262	IsSeparator	Check if a Unicode character is a separator character.
258	IsSurrogate	Check whether a Unicode character is a surrogate.
260	IsSurrogatePair	Check if a pair of characters is a set of high/low surrogate characters.
263	IsSymbol	Check if a Unicode character is a symbol character.
263	IsUpper	Check whether a Unicode character is an uppercase letter.
263	IsWhiteSpace	Check whether a Unicode character is a whitespace character.
264	ToLower	Convert a character or string to lowercase.
264	ToUpper	Convert a character or string to uppercase.

2.5.3 TCharacter.Create

Synopsis: Constructor (do not call).

Declaration: `constructor Create`

Visibility: public

Description: `Create` is provided for completeness and Delphi compatibility, but should not be called in FPC code, it will raise an exception.

Errors: Any attempt to call `Create` will result in an exception being raised.

2.5.4 TCharacter.ConvertFromUtf32

Synopsis: Convert a UTF32 character to UnicodeString.

Declaration:
```
class function ConvertFromUtf32(AChar: UCS4Char) : UnicodeString
                                ;  Static
```

Visibility: public

Description: `TCharacter.ConvertFromUtf32` converts a single UTF32 character `AChar` to a UTF16 string. This is the opposite of TCharacter.ConvertToUtf32 (256).

The result is a string, since multiple UTF16 characters can be needed to encode a single UTF32 character.

Errors: If `AChar` is not in the valid range of UTF32 characters, an EArgumentOutOfRangeException (247) exception is raised.

See also: EArgumentOutOfRangeException (247), TCharacter.ConvertToUtf32 (256)

2.5.5 TCharacter.ConvertToUtf32

Synopsis: Convert a UTF16 character to a UTF32 character.

Declaration:
```
class function ConvertToUtf32(const AString: UnicodeString;
                              AIndex: Integer) : UCS4Char;  Overload
                              ;  Static
class function ConvertToUtf32(const AString: UnicodeString;
                              AIndex: Integer; out ACharLength: Integer)
                               : UCS4Char;  Overload;  Static
class function ConvertToUtf32(const AHighSurrogate: UnicodeChar;
                              const ALowSurrogate: UnicodeChar)
                               : UCS4Char;  Overload;  Static
```

Visibility: public

Description: `TCharacter.ConvertToUtf32` converts a UTF16-encoded Unicode character to a Unicode32 character. This is the opposite of TCharacter.ConvertFromUtf32 (256). The function exists in several overloaded versions, to be able to present the Unicode character in one of 2 ways:

1. As a position `AIndex` (in unicodechar units) in a string `AString` to a Unicode32 character. The source is a string, since multiple UTF16 characters can be needed to encode a single UTF32 character. In this form, Optionally, the character length (1 or 2) can be returned in `ACharLength`.
2. As 2 UTF16 Unicode characters, representing the high and low surrogate pairs: `AHighSurrogate` and `ALowSurrogate`.

Errors: If AIndex is not a valid character index in the string AString, an EArgumentOutOfRangeException (247) exception is raised. If the character at that position is not complete, an EArgumentException (247) exception is raised.

See also: TCharacter.ConvertFromUtf32 (256), EArgumentOutOfRangeException (247), EArgumentException (247)

2.5.6 TCharacter.GetNumericValue

Synopsis: Get the numeric value of the character.

Declaration:
```
class function GetNumericValue(AChar: UnicodeChar) : Double;  Overload
                               ;  Static
class function GetNumericValue(const AString: UnicodeString;
                               AIndex: Integer) : Double;  Overload
                               ;  Static
```

Visibility: public

Description: TCharacter.GetNumericValue returns the numerical value (ID) of the Unicode character. The character can be presented in 2 ways: AChar, a UTF16 Unicode character, or a surrogate pair in a Unicode string AString starting at position AIndex.

Errors: If AIndex is not a valid character index in the string AString, an EArgumentOutOfRangeException (247) exception is raised. If the character at that position is not complete, an EArgumentException (247) exception is raised.

See also: TCharacter.GetUnicodeCategory (257)

2.5.7 TCharacter.GetUnicodeCategory

Synopsis: Get the Unicode category of a character.

Declaration:
```
class function GetUnicodeCategory(AChar: UnicodeChar) : TUnicodeCategory
                                  ;  Overload;  Static
class function GetUnicodeCategory(const AString: UnicodeString;
                                  AIndex: Integer) : TUnicodeCategory
                                  ;  Overload;  Static
```

Visibility: public

Description: TCharacter.GetUnicodeCategory returns the Unicode category of a character. The character can be presented in 2 ways: AChar, a UTF16 Unicode character, or a surrogate pair in a Unicode string AString starting at position AIndex.

Errors: If AIndex is not a valid character index in the string AString, an EArgumentOutOfRangeException (247) exception is raised. If the character at that position is not complete, an EArgumentException (247) exception is raised.

See also: TUnicodeCategory (248)

2.5.8 TCharacter.IsControl

Synopsis: Check whether a Unicode character is a Unicode control character.

Declaration:
```
class function IsControl(AChar: UnicodeChar) : Boolean;  Overload
                              ;  Static
class function IsControl(const AString: UnicodeString; AIndex: Integer)
                               : Boolean;  Overload;  Static
```

Visibility: public

Description: `IsControl` returns `True` if a Unicode character has category `ucControl`. The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (247) exception is raised. If the character at that position is not complete, an EArgumentException (247) exception is raised.

See also: GetUnicodeCategory (257), IsDigit (258), IsSurrogate (258), IsHighSurrogate (259), IsLowSurrogate (259), IsSurrogatePair (260), IsLetter (260), IsLetterOrDigit (260), IsLower (261), IsNumber (261), IsPunctuation (262), IsSeparator (262), IsSymbol (263), IsUpper (263), IsWhiteSpace (263)

2.5.9 TCharacter.IsDigit

Synopsis: Check whether a Unicode character is a digit.

Declaration:
```
class function IsDigit(AChar: UnicodeChar) : Boolean;  Overload;  Static
class function IsDigit(const AString: UnicodeString; AIndex: Integer)
                        : Boolean;  Overload;  Static
```

Visibility: public

Description: `IsDigit` returns `True` if a Unicode character has category `ucDecimalNumber`. The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (247) exception is raised. If the character at that position is not complete, an EArgumentException (247) exception is raised.

See also: IsControl (257), IsDigit (258), IsSurrogate (258), IsHighSurrogate (259), IsLowSurrogate (259), IsSurrogatePair (260), IsLetter (260), IsLetterOrDigit (260), IsLower (261), IsNumber (261), IsPunctuation (262), IsSeparator (262), IsSymbol (263), IsUpper (263), IsWhiteSpace (263)

2.5.10 TCharacter.IsSurrogate

Synopsis: Check whether a Unicode character is a surrogate.

Declaration:
```
class function IsSurrogate(AChar: UnicodeChar) : Boolean;  Overload
                            ;  Static
class function IsSurrogate(const AString: UnicodeString;
                            AIndex: Integer) : Boolean;  Overload;  Static
```

Visibility: public

Description: `IsSurrogate` returns `True` if a Unicode character has category `ucSurrogate`. The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If AIndex is not a valid character index in the string AString, an EArgumentOutOfRangeException (247) exception is raised. If the character at that position is not complete, an EArgumentException (247) exception is raised.

See also: EArgumentException (247), IsControl (257), IsDigit (258), IsHighSurrogate (259), IsLowSurrogate (259), IsLetter (260), IsLetterOrDigit (260), IsLower (261), IsNumber (261), IsPunctuation (262), IsSymbol (263), IsUpper (263), IsWhiteSpace (263)

2.5.11 TCharacter.IsHighSurrogate

Synopsis: Check whether a Unicode character is a surrogate in the high range.

Declaration:
```
class function IsHighSurrogate(AChar: UnicodeChar) : Boolean;  Overload
                               ;  Static
class function IsHighSurrogate(const AString: UnicodeString;
                               AIndex: Integer) : Boolean;  Overload
                               ;  Static
```

Visibility: public

Description: IsHighSurrogate returns True if a Unicode character has category ucSurrogate and is in the high range of the surrogate characters (between HIGH_SURROGATE_BEGIN and HIGH_SURROGATE_END). The character can be specified as a UTF16 character AChar or a UTF16 encoded character starting at position AIndex in string AString.

Errors: If AIndex is not a valid character index in the string AString, an EArgumentOutOfRangeException (247) exception is raised. If the character at that position is not complete, an EArgumentException (247) exception is raised.

See also: EArgumentException (247), IsControl (257), IsDigit (258), IsSurrogate (258), IsLowSurrogate (259), IsLetter (260), IsLetterOrDigit (260), IsLower (261), IsNumber (261), IsPunctuation (262), IsSymbol (263), IsUpper (263), IsWhiteSpace (263)

2.5.12 TCharacter.IsLowSurrogate

Synopsis: Check whether a Unicode character is a surrogate in the low range.

Declaration:
```
class function IsLowSurrogate(AChar: UnicodeChar) : Boolean;  Overload
                              ;  Static
class function IsLowSurrogate(const AString: UnicodeString;
                              AIndex: Integer) : Boolean;  Overload
                              ;  Static
```

Visibility: public

Description: IsLowSurrogate returns True if a Unicode character has category ucSurrogate and is in the low range of the surrogate characters (between LOW_SURROGATE_BEGIN and LOW_SURROGATE_END). The character can be specified as a UTF16 character AChar or a UTF16 encoded character starting at position AIndex in string AString.

Errors: If AIndex is not a valid character index in the string AString, an EArgumentOutOfRangeException (247) exception is raised. If the character at that position is not complete, an EArgumentException (247) exception is raised.

See also: EArgumentException (247), IsControl (257), IsDigit (258), IsSurrogate (258), IsHighSurrogate (259), IsLetter (260), IsLetterOrDigit (260), IsLower (261), IsNumber (261), IsPunctuation (262), IsSymbol (263), IsUpper (263), IsWhiteSpace (263)

2.5.13 TCharacter.IsSurrogatePair

Synopsis: Check if a pair of characters is a set of high/low surrogate characters.

Declaration:
```
class function IsSurrogatePair(const AHighSurrogate: UnicodeChar;
                               const ALowSurrogate: UnicodeChar)
                                : Boolean;  Overload;  Static
class function IsSurrogatePair(const AString: UnicodeString;
                               AIndex: Integer) : Boolean;  Overload
                               ;  Static
```

Visibility: public

Description: `IsSurrogatePair` returns `True` if `AHighSurrogate` and `ALowSurrogate` form a valid Unicode surrogate pair. (`AHighSurrogate` is a high surrogate and `ALowSurrogate` a matching low surrogate) The character can be specified as a UTF16 character `AChar` or a pair of UTF16 encoded characters starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (247) exception is raised. If the character at that position is not complete, an EArgumentException (247) exception is raised.

See also: EArgumentException (247), IsControl (257), IsDigit (258), IsSurrogate (258), IsHighSurrogate (259), IsLowSurrogate (259), IsLetter (260), IsLetterOrDigit (260), IsLower (261), IsNumber (261), IsPunctuation (262), IsSymbol (263), IsUpper (263), IsWhiteSpace (263)

2.5.14 TCharacter.IsLetter

Synopsis: Check if a Unicode character is a letter.

Declaration:
```
class function IsLetter(AChar: UnicodeChar) : Boolean;  Overload
                        ;  Static
class function IsLetter(const AString: UnicodeString; AIndex: Integer)
                        : Boolean;  Overload;  Static
```

Visibility: public

Description: `IsLetter` returns `True` if a Unicode character has category that is one of the letter categories (`ucUppercaseLetter`, `ucLowercaseLetter`, `ucTitlecaseLetter`, `ucModifierLetter`, `ucOtherLetter`). The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (247) exception is raised. If the character at that position is not complete, an

See also: EArgumentException (247), IsControl (257), IsDigit (258), IsSurrogate (258), IsHighSurrogate (259), IsLowSurrogate (259), IsSurrogatePair (260), IsLetter (260), IsLetterOrDigit (260), IsLower (261), IsNumber (261), IsPunctuation (262), IsSymbol (263), IsUpper (263), IsWhiteSpace (263)

2.5.15 TCharacter.IsLetterOrDigit

Synopsis: Check if a Unicode character is a letter or digit.

Declaration:
```
class function IsLetterOrDigit(AChar: UnicodeChar) : Boolean;  Overload
                               ;  Static
class function IsLetterOrDigit(const AString: UnicodeString;
                               AIndex: Integer) : Boolean;  Overload
                               ;  Static
```

Visibility: public

Description: `IsLetterOrDigit` returns `True` if a Unicode character has category that is one of the letter categories (`ucUppercaseLetter`, `ucLowercaseLetter`, `ucTitlecaseLetter`, `ucModifierLetter`, `ucOtherLetter`,`ucDecimalNumber`,`ucLetterNumber`). The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (247) exception is raised. If the character at that position is not complete, an EArgumentException (247) exception is raised.

See also: IsControl (257), IsDigit (258), IsSurrogate (258), IsHighSurrogate (259), IsLowSurrogate (259), IsSurrogatePair (260), IsLetter (260), IsLower (261), IsNumber (261), IsPunctuation (262), IsSymbol (263), IsUpper (263), IsWhiteSpace (263)

2.5.16 TCharacter.IsLower

Synopsis: Check if a Unicode character is a lowercase letter.

Declaration:
```
class function IsLower(AChar: UnicodeChar) : Boolean; Overload; Static
class function IsLower(const AString: UnicodeString; AIndex: Integer)
                       : Boolean; Overload; Static
```

Visibility: public

Description: `IsLower` returns `True` if a Unicode character has category `ucLowercaseLetter`. The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (247) exception is raised. If the character at that position is not complete, an EArgumentException (247) exception is raised.

See also: IsControl (257), IsDigit (258), IsSurrogate (258), IsHighSurrogate (259), IsLowSurrogate (259), IsSurrogatePair (260), IsLetter (260), IsLetterOrDigit (260), IsNumber (261), IsPunctuation (262), IsSymbol (263), IsUpper (263), IsWhiteSpace (263)

2.5.17 TCharacter.IsNumber

Synopsis: Check if a Unicode character is a number.

Declaration:
```
class function IsNumber(AChar: UnicodeChar) : Boolean; Overload
                        ; Static
class function IsNumber(const AString: UnicodeString; AIndex: Integer)
                        : Boolean; Overload; Static
```

Visibility: public

Description: `IsNumber` returns `True` if a Unicode character has category that is one of the number categories (`ucDecimalNumber`, `ucLetterNumber`, `ucOtherNumber`). The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (247) exception is raised. If the character at that position is not complete, an EArgumentException (247) exception is raised.

See also: IsControl (257), IsDigit (258), IsSurrogate (258), IsHighSurrogate (259), IsLowSurrogate (259), IsSurrogatePair (260), IsLetter (260), IsLetterOrDigit (260), IsLower (261), IsNumber (261), IsPunctuation (262), IsSymbol (263), IsUpper (263), IsWhiteSpace (263)

2.5.18 TCharacter.IsPunctuation

Synopsis: Check if a Unicode character is a punctuation character.

Declaration:
```
class function IsPunctuation(AChar: UnicodeChar) : Boolean;  Overload
                            ;  Static
class function IsPunctuation(const AString: UnicodeString;
                            AIndex: Integer) : Boolean;  Overload
                            ;  Static
```

Visibility: public

Description: `IsPunctuation` returns `True` if a Unicode character has category that is one of the punctuation categories (`ucConnectPunctuation`, `ucDashPunctuation`, `ucOpenPunctuation`, `ucClosePunctuation`, `ucInitialPunctuation`, `ucFinalPunctuation`, `ucOtherPunctuation`). The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (247) exception is raised. If the character at that position is not complete, an EArgumentException (247) exception is raised.

See also: IsControl (257), IsDigit (258), IsSurrogate (258), IsHighSurrogate (259), IsLowSurrogate (259), IsSurrogatePair (260), IsLetter (260), IsLetterOrDigit (260), IsLower (261), IsNumber (261), IsSymbol (263), IsUpper (263), IsWhiteSpace (263)

2.5.19 TCharacter.IsSeparator

Synopsis: Check if a Unicode character is a separator character.

Declaration:
```
class function IsSeparator(AChar: UnicodeChar) : Boolean;  Overload
                          ;  Static
class function IsSeparator(const AString: UnicodeString;
                          AIndex: Integer) : Boolean;  Overload;  Static
```

Visibility: public

Description: `IsSeparator` returns `True` if a Unicode character has category that is one of the separator categories (`ucSpaceSeparator`, `ucLineSeparator`, `ucParagraphSeparator`). The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (247) exception is raised. If the character at that position is not complete, an EArgumentException (247) exception is raised.

See also: IsControl (257), IsDigit (258), IsSurrogate (258), IsHighSurrogate (259), IsLowSurrogate (259), IsSurrogatePair (260), IsLetter (260), IsLetterOrDigit (260), IsLower (261), IsNumber (261), IsPunctuation (262), IsSymbol (263), IsUpper (263), IsWhiteSpace (263)

2.5.20 TCharacter.IsSymbol

Synopsis: Check if a Unicode character is a symbol character.

Declaration:
```
class function IsSymbol(AChar: UnicodeChar) : Boolean;  Overload
                        ;  Static
class function IsSymbol(const AString: UnicodeString; AIndex: Integer)
                        : Boolean;  Overload;  Static
```

Visibility: public

Description: `IsSymbol` returns `True` if a Unicode character has category that is one of the symbol categories (`ucMathSymbol`, `ucCurrencySymbol`, `ucModifierSymbol`, `ucOtherSymbol`). The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (247) exception is raised. If the character at that position is not complete, an EArgumentException (247) exception is raised.

See also: IsControl (257), IsDigit (258), IsSurrogate (258), IsHighSurrogate (259), IsLowSurrogate (259), IsSurrogatePair (260), IsLetter (260), IsLetterOrDigit (260), IsLower (261), IsNumber (261), IsPunctuation (262), IsSeparator (262), IsUpper (263), IsWhiteSpace (263)

2.5.21 TCharacter.IsUpper

Synopsis: Check whether a Unicode character is an uppercase letter.

Declaration:
```
class function IsUpper(AChar: UnicodeChar) : Boolean;  Overload;  Static
class function IsUpper(const AString: UnicodeString; AIndex: Integer)
                       : Boolean;  Overload;  Static
```

Visibility: public

Description: `IsUpper` returns `True` if a Unicode character has category `ucUppercaseLetter`. The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (247) exception is raised. If the character at that position is not complete, an EArgumentException (247) exception is raised.

See also: IsControl (257), IsDigit (258), IsSurrogate (258), IsHighSurrogate (259), IsLowSurrogate (259), IsSurrogatePair (260), IsLetter (260), IsLetterOrDigit (260), IsLower (261), IsNumber (261), IsPunctuation (262), IsSeparator (262), IsSymbol (263), IsWhiteSpace (263)

2.5.22 TCharacter.IsWhiteSpace

Synopsis: Check whether a Unicode character is a whitespace character.

Declaration:
```
class function IsWhiteSpace(AChar: UnicodeChar) : Boolean;  Overload
                            ;  Static
class function IsWhiteSpace(const AString: UnicodeString;
                            AIndex: Integer) : Boolean;  Overload
                            ;  Static
```

Visibility: public

Description: `IsUpper` returns `True` if a Unicode character has is a whitespace character. It checks the character properties. The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an EArgumentOutOfRangeException (247) exception is raised. If the character at that position is not complete, an EArgumentException (247) exception is raised.

See also: IsControl (257), IsDigit (258), IsSurrogate (258), IsHighSurrogate (259), IsLowSurrogate (259), IsSurrogatePair (260), IsLetter (260), IsLetterOrDigit (260), IsLower (261), IsNumber (261), IsPunctuation (262), IsSeparator (262), IsSymbol (263), IsUpper (263)

2.5.23 TCharacter.ToLower

Synopsis: Convert a character or string to lowercase.

Declaration:
```
class function ToLower(AChar: UnicodeChar) : UnicodeChar;  Overload
                       ;  Static
class function ToLower(const AString: UnicodeString) : UnicodeString
                       ;  Overload;  Static
class function ToLower(const AString: UnicodeString;
                       const AOptions: TCharacterOptions) : UnicodeString
                       ;  Overload;  Static
```

Visibility: public

Description: `ToLower` converts the Unicode character `AChar` or string `AString` to lowercase. `Options` determines the behaviour of the conversion: if `AOptions` contains `coIgnoreInvalidSequence` then no exception will be raised when the string or character contains an invalid Unicode sequence. The default behaviour is to raise an EArgumentException (247) exception when this happens.

Errors: If an invalid character is encountered, an EArgumentException (247) exception is raised, unless `coIgnoreInvalidSequence` is specified in the options.

See also: TCharacter.ToUpper (264), TCharacter.IsLower (261), TCharacter.IsUpper (263)

2.5.24 TCharacter.ToUpper

Synopsis: Convert a character or string to uppercase.

Declaration:
```
class function ToUpper(AChar: UnicodeChar) : UnicodeChar;  Overload
                       ;  Static
class function ToUpper(const AString: UnicodeString) : UnicodeString
                       ;  Overload;  Static
class function ToUpper(const AString: UnicodeString;
                       const AOptions: TCharacterOptions) : UnicodeString
                       ;  Overload;  Static
```

Visibility: public

Description: `ToUpper` converts the Unicode character `AChar` or string `AString` to uppercase. `Options` determines the behaviour of the conversion: if `AOptions` contains `coIgnoreInvalidSequence` then no exception will be raised when the string or character contains an invalid Unicode sequence. The default behaviour is to raise an EArgumentException (247) exception when this happens.

Errors: If an invalid character is encountered, an EArgumentException (247) exception is raised, unless `coIgnoreInvalidSequence` is specified in the options.

See also: TCharacter.ToUpper (264), TCharacter.IsLower (261), TCharacter.IsUpper (263)

Chapter 3

Reference for unit 'charset'

3.1 Used units

Table 3.1: Used units by unit 'charset'

Name	Page
System	1362

3.2 Overview

The charset unit can be used to load single-byte character set (code page) descriptions. It is used in the fpwidestring (266) unit to add support for converting single-byte codepage strings to Unicode strings (and vice versa).

Data of a code page may be included using one of the ready-made units, or can be loaded (in a binary form) at runtime with the loadbinaryunicodemapping (268) function. The binary files have the .bcm extension and are produced by the creumap utility distributed with Free Pascal.

Pre-made units are available for the following codepages: cp895 (266), cp932 (266), cp936 (266), cp949 (266) and cp950 (266).

3.3 Constants, types and variables

3.3.1 Constants

```
BINARY_MAPPING_FILE_EXT = '.bcm'
```

`BINARY_MAPPING_FILE_EXT` contains the default extension of a file containing a binary-coded map.

3.3.2 Types

```
preversecharmapping = ^treversecharmapping
```

Pointer to `treversecharmapping`.

```
punicodecharmapping = ^tunicodecharmapping
```

Pointer to `tunicodecharmapping`.

```
punicodemap = ^tunicodemap
```

Pointer to `tunicodemap`.

```
tunicodechar = Word
```

`tunicodechar` is a type used to represent Unicode characters in this file, it should not be used for other Unicode routines.

```
tunicodecharmappingflag = (umf_noinfo,umf_leadbyte,umf_undefined,
  umf_unused)
```

Table 3.2: Enumeration values for type tunicodecharmappingflag

Value	Explanation
umf_leadbyte	Unicode character uses leading byte.
umf_noinfo	No extra information about Unicode character.
umf_undefined	Currently unused.
umf_unused	Unused position in code page.

`tunicodecharmappingflag` contains various Flags describing information about a Unicode character.

```
tunicodestring = ^tunicodechar
```

`tunicodestring` is a type used to represent Unicode strings in this file, it should not be used for other Unicode routines.

3.4 Procedures and functions

3.4.1 getascii

Synopsis: Convert Unicode character or string to single-byte character or string.

Declaration:
```
function getascii(c: tunicodechar; p: punicodemap) : string
function getascii(c: tunicodechar; p: punicodemap; ABuffer: PAnsiChar;
                  ABufferLen: LongInt) : LongInt
```

Visibility: default

Description: `getascii` converts a Unicode character `c` to one or more single-byte characters according to the map in `p`. The result can be a string containing up to 2 characters, or the number of characters copied to the buffer `ABuffer` with length `ABufferLen`.

Errors: If the character cannot be translated, ASCII character 63 is returned (or copied to the buffer). In the case of the buffer variant of the function, -1 is then returned. If the buffer is not large enough, -1 is returned.

See also: getunicode (268)

3.4.2 getmap

Synopsis: Find a codepage map.

Declaration:
```
function getmap(const s: string) : punicodemap
function getmap(cp: Word) : punicodemap
```

Visibility: default

Description: `getmap` looks in the registered codepage mappings and returns the mapping for the requested codepage. The codepage can be specified using a name `s` or a numerical identifier `cp`. The search is case sensitive.

Errors: if the requested map is not found, `Nil` is returned.

See also: registermapping (270), registerbinarymapping (269), mappingavailable (269)

3.4.3 getunicode

Synopsis: Map single-byte character to Unicode character.

Declaration:
```
function getunicode(c: Char; p: punicodemap) : tunicodechar
function getunicode(AAnsiStr: PAnsiChar; AAnsiLen: LongInt;
                    AMap: punicodemap; ADest: tunicodestring) : LongInt
```

Visibility: default

Description: The first form of `getunicode` will map a single character `c` to its Unicode equivalent for mapping `p`. If no equivalent can be found, `0` is returned.

The second form of `getunicode` will transform a string (specified using a pointer `AAnsiStr` to a buffer with length `AAnsiLen`) to a Unicode string using single byte codepage map `AMap`. It returns the number of Unicode characters. If `ADest` is `Nil` then just the number of characters is returned. If `ADest` is not `nil`, it must point to a buffer large enough to contain the Unicode string, and the converted string will be copied to it.

Errors: No checking on the validity of the buffers is done.

See also: getascii (267)

3.4.4 loadbinaryunicodemapping

Synopsis: Load binary single-byte codepage to Unicode map from file or memory.

Declaration:
```
function loadbinaryunicodemapping(const directory: string;
                                  const cpname: string) : punicodemap
                                  ;  Overload
function loadbinaryunicodemapping(const filename: string) : punicodemap
                                  ;  Overload
function loadbinaryunicodemapping(const AData: Pointer;
                                  const ADataLength: Integer)
                                   : punicodemap;  Overload
```

Visibility: default

Description: `loadbinaryunicodemapping` loads a binary description of a single-byte Unicode mapping. The mapping can reside in a file, in which case the file to load can be specified using a filename `filename` or using a directory `directory` and codepage name `cpname`. In the latter case, a

suffix consisting of _le or _be depending on the endianness of the current platform will be appended, and the filename extension is .bcm. Note that the file names may be case sensitive.

The data can also be loaded from a memory block `AData` with size `ADataLength`.

It will produce an in-memory map of the file. It returns a pointer to the map, or `Nil` if something went wrong. The resulting mapping can be registered using registermapping (270).

Errors: On error, `Nil` is returned.

See also: loadunicodemapping (269), registermapping (270)

3.4.5 loadunicodemapping

Synopsis: Load textual single-byte codepage to Unicode map from file.

Declaration:
```
function loadunicodemapping(const cpname: string; const f: string;
                            cp: Word) : punicodemap
```

Visibility: default

Description: `loadunicodemapping` loads a text description of a single-byte Unicode mapping. It will analyse the textual description in file `f`, and produce an in-memory map of the file. It returns a pointer to the map, or `Nil` if something went wrong. The Unicode map name must be specified in `cpName`, and the numerical identifier in `cp`

The resulting mapping can be registered using registermapping (270).

Errors: On error, `Nil` is returned.

See also: loadbinaryunicodemapping (268), registermapping (270)

3.4.6 mappingavailable

Synopsis: Check if a mapping is available for a specified code page.

Declaration:
```
function mappingavailable(const s: string) : Boolean
function mappingavailable(cp: Word) : Boolean
```

Visibility: default

Description: `mappingavailable` returns `True` if a mapping for a specified code page (using name `s` or numerical identifier `cp`) is available, or `False` if it is not.

See also: registermapping (270), registerbinarymapping (269), getmap (268)

3.4.7 registerbinarymapping

Synopsis: Load and register binary single-byte codepage to Unicode map from file.

Declaration:
```
function registerbinarymapping(const directory: string;
                               const cpname: string) : Boolean
```

Visibility: default

Description: `registerbinarymapping` calls loadbinaryunicodemapping (268) using `directory` and `cpname` and registers the resulting mapping, if any was successfully loaded, using registermapping (270). It returns `True` if the operation was successful.

Errors: On error, `False` is returned.

See also: loadbinaryunicodemapping (268), registermapping (270)

3.4.8 registermapping

Synopsis: Register mapping.

Declaration: `procedure registermapping(p: punicodemap)`

Visibility: default

Description: `RegisterMapping` registers mapping `p` in the registry of single-byte codepages. No attempt is made to avoid double registrations. In case of doubles, the last registered mapping will be used first.

See also: loadunicodemapping (269), loadbinaryunicodemapping (268), registerbinarymapping (269)

3.5 treversecharmapping

```
treversecharmapping = packed record
  unicode : tunicodechar;
  char1
  : Byte;
  char2 : Byte;
end
```

`treversecharmapping` describes how a Unicode character can be created in terms of single-byte characters.

3.6 TSerializedMapHeader

```
TSerializedMapHeader = packed record
  cpName : string;
  cp : UInt16
  ;
  mapLength : UInt32;
  lastChar : Int32;
  reverseMapLength : UInt32
  ;
end
```

`TSerializedMapHeader` is a record describing the binary map data file. The contents of this record can be found at offset zero of a (.bcm) file containing a single-byte Unicode map.

3.7 tunicodecharmapping

```
tunicodecharmapping = packed record
  unicode : tunicodechar;
  flag
  : tunicodecharmappingflag;
  reserved : Byte;
end
```

`tunicodecharmapping` describes a Unicode character. An array of these mappings is built for each character in the single-byte character set,

3.8 tunicodemap

```
tunicodemap = record
  cpname : string;
  cp : Word;
  map : punicodecharmapping
  ;
  lastchar : LongInt;
  reversemap : preversecharmapping;
  reversemaplength
  : LongInt;
  next : punicodemap;
  internalmap : Boolean;
end
```

`tunicodemap` describes a complete mapping between a single-byte code page and a Unicode character set. It contains both a forward and backward mapping.

Chapter 4

Reference for unit 'Classes'

4.1 Used units

Table 4.1: Used units by unit 'Classes'

Name	Page
rtlconsts	??
System	1362
sysutils	1635
Types	1965
TypInfo	2016

4.2 Overview

This documentation describes the FPC classes unit. The Classes unit contains basic classes for the Free Component Library (FCL):

- a TList (415) class for maintaining lists of pointers,
- TStringList (470) for lists of strings,
- TCollection (365) to manage collections of objects
- TStream (455) classes to support streaming.

Furthermore it introduces methods for object persistence, and classes that understand an owner-owned relationship, with automatic memory management.

4.3 Constants, types and variables

4.3.1 Constants

```
BITSHIFT = 5
```

Used to calculate the size of a bits array.

```
dupAccept = Types.dupAccept
```

Duplicate values can be added to the list.

```
dupError = Types.dupError
```

If an attempt is made to add a duplicate value to the list, an EStringListError (312) exception is raised.

```
dupIgnore = Types.dupIgnore
```

Duplicate values will not be added to the list, but no error will be triggered.

```
FilerSignature : Array[1..4] of Char = 'TPF0'
```

Constant that is found at the start of a binary stream containing a streamed component.

```
fmCreate = $FF00
```

TFileStream.Create (396) creates a new file if needed.

```
fmOpenRead = 0
```

TFileStream.Create (396) opens a file with read-only access.

```
fmOpenReadWrite = 2
```

TFileStream.Create (396) opens a file with read-write access.

```
fmOpenWrite = 1
```

TFileStream.Create (396) opens a file with write-only access.

```
MASK = 31
```

Bitmask with all bits on.

```
MaxBitFlags = $7FFFFFE0
```

Maximum number of bits in TBits collection.

```
MaxBitRec = MaxBitFlags div SizeOf(cardinal) * 8
```

Maximum number of bit records in TBits.

```
MaxListSize = Maxint div 16
```

This constant sets the maximum number of elements in a TList (415).

```
scAlt = $8000
```

Indicates ALT key in a keyboard shortcut.

```
scCtrl = $4000
```

indicates CTRL key in a keyboard shortcut.

```
scNone = 0
```

Indicates no special key is pressed in a keyboard shortcut.

```
scShift = $2000
```

Indicates Shift key in a keyboard shortcut.

```
SGUIDObserved = '{663C603C-3F3C-4CC5-823C-AC8079F979E5}'
```

Observed interface GUID as a string.

```
SGUIDObserver = '{BC7376EA-199C-4C2A-8684-F4805F0691CA}'
```

Observer interface GUID as a string.

```
soFromBeginning = 0
```

Seek (457) starts relative to the stream origin.

```
soFromCurrent = 1
```

Seek (457) starts relative to the current position in the stream.

```
soFromEnd = 2
```

Seek (457) starts relative to the stream end.

```
toEOF = Char(0)
```

Value returned by TParser.Token (435) when the end of the input stream was reached.

```
toFloat = Char(4)
```

Value returned by TParser.Token (435) when a floating point value was found in the input stream.

```
toInteger = Char(3)
```

Value returned by TParser.Token (435) when an integer was found in the input stream.

```
toString = Char(2)
```

Value returned by TParser.Token (435) when a string was found in the input stream.

```
toSymbol = Char(1)
```

Value returned by TParser.Token (435) when a symbol was found in the input stream.

```
toWString = Char(5)
```

Value returned by TParser.Token (435) when a widestring was found in the input stream.

4.3.2 Types

```
EListError = SysUtils.EListError
```

If an error occurs in one of the TList (415) or TStrings (475) methods, then a `EListError` exception is raised. This can occur in one of the following cases:

1. There is not enough memory to expand the list.
2. The list tried to grow beyond its maximal capacity.
3. An attempt was made to reduce the capacity of the list below the current element count.
4. An attempt was made to set the list count to a negative value.
5. A non-existent element of the list was referenced. (i.e. the list index was out of bounds)
6. An attempt was made to move an item to a position outside the list's bounds.

```
HMODULE = PtrInt
```

FPC doesn't support modules yet, so this is a dummy type.

```
HRSRC = TFPResourceHandle deprecated
```

This type is provided for Delphi compatibility, it is used for resource streams.

```
PPointerList = ^TPointerList
```

Pointer to an array of pointers.

```
PStringItem = ^TStringItem
```

Pointer to a TStringItem (309) record.

```
PStringItemList = ^TStringItemList
```

Pointer to a TStringItemList (287).

```
TActionEvent = procedure(Action: TBasicAction; var Handled: Boolean
  )
                           of object


TActiveXRegType = (axrComponentOnly,axrIncludeDescendants)
```

Table 4.2: Enumeration values for type TActiveXRegType

Value	Explanation
axrComponentOnly	
axrIncludeDescendants	

This type is provided for compatibility only, and is currently not used in Free Pascal.

```
TAlignment = (taLeftJustify,taRightJustify,taCenter)
```

Table 4.3: Enumeration values for type TAlignment

Value	Explanation
taCenter	Text is displayed centered.
taLeftJustify	Text is displayed aligned to the left.
taRightJustify	Text is displayed aligned to the right.

The `TAlignment` type is used to specify the alignment of the text in controls that display a text.

```
TAncestorNotFoundEvent = procedure(Reader: TReader;
  const ComponentName: string;
  ComponentClass: TPersistentClass;
  var Component: TComponent) of object
```

This event occurs when an ancestor component cannot be found.

```
TBasicActionClass = Class of TBasicAction
```

TBasicAction (339) class reference.

```
TBasicActionLinkClass = Class of TBasicActionLink
```

TBasicActionLink (343) class reference.

```
TBiDiMode = (bdLeftToRight,bdRightToLeft,bdRightToLeftNoAlign,
  bdRightToLeftReadingOnly)
```

Table 4.4: Enumeration values for type TBiDiMode

Value	Explanation
bdLeftToRight	Texts read from left to right.
bdRightToLeft	Texts read from right to left.
bdRightToLeftNoAlign	Texts read from right to left, but not right-aligned.
bdRightToLeftReadingOnly	Texts read from right to left.

`TBiDiMode` describes bi-directional support for displaying texts.

```
TBitArray = Array[0..MaxBitRec-1] of Cardinal
```

Array to store bits.

```
TCollectionItemClass = Class of TCollectionItem
```

`TCollectionItemClass` is used by the TCollection.ItemClass (371) property of TCollection (365) to identify the descendant class of TCollectionItem (373) which should be created and managed.

```
TCollectionNotification = (cnAdded,cnExtracting,cnDeleting)
```

Table 4.5: Enumeration values for type TCollectionNotification

Value	Explanation
cnAdded	An item is added to the collection.
cnDeleting	An item is deleted from the collection.
cnExtracting	An item is extracted from the collection.

`TCollectionNotification` is used in the TCollection (365) class to send notifications about changes to the collection.

```
TCollectionSortCompare = function(Item1: TCollectionItem;
  Item2: TCollectionItem) : Integer
```

`TCollectionSortCompare` is the prototype for a callback used in the TCollection.Sort (370) method. The procedure should compare `Item1` and `Item2` and return an integer:

Result < 0 if `Item1` comes before `Item2`

Result = 0 if `Item1` is at the same level as `Item2`

Result > 0 if `Item1` comes after `Item2`

```
TComponentClass = Class of TComponent
```

The `TComponentClass` type is used when constructing TComponent (376) descendant instances and when registering components.

```
TComponentName = String
```

Names of components are of type `TComponentName`. By specifying a different type, the Object inspector can handle this property differently than a standard string property.

```
TComponentState= Set of (csLoading,csReading,csWriting,csDestroying
  ,
                         csDesigning,csAncestor,csUpdating,csFixups
  ,
                         csFreeNotification,csInline,csDesignInstance
  )
```

Table 4.6: Enumeration values for type

Value	Explanation
csAncestor	The component is being streamed as part of a frame (?).
csDesigning	The component is being designed in an IDE.
csDesignInstance	??
csDestroying	The component is being destroyed.
csFixups	The component's references to other components are being fixed.
csFreeNotification	Indicates whether the component has freenotifications.
csInline	Component is part of a frame (?).
csLoading	The component is being loaded from the stream.
csReading	Properties are being read from the stream.
csUpdating	The component is being updated. This value is provided for Delphi compatibility and not used in FPC.
csWriting	Properties are being written to the stream.

The following values are possible:

csLoading The component (and all child components) are being loaded from a stream. This means that a TReader (440) instance is reading properties from this and child components from a stream and is applying the values found in the stream to the properties.

csReading The properties of this component are being read from a stream. This means that a TReader (440) instance is reading properties from this component from a stream and is applying the values.

csWriting The properties of this component are being written to a stream. This means that a TWriter (524) instance is writing properties from this component to a stream.

csDestroying The component is being destroyed.

csDesigning The component is being designed in an IDE.

csAncestor The component has a design ancestor. This is used to record differences between a component and it's design ancestor. For example a form `TForm2` inherited from a form `TForm1`. `TForm1` and all its components are copied to `TForm2`. `TForm2` and all its inherited components have `csAncestor` set. Only differences between `TForm1` and `TForm2` are stored in the stream of `TForm2`. The child components of a frame put onto a form have `csAncestor` too.

csInline The component is a nested top level component. For example a frame on a form. The children of the frame do not have `csInline`, unless they are other frames.

csDesignInstance The component is designed (`csDesigning`) and is a root component, meaning it has no owner (`Owner=nil`).

csFixups The component's references to other components are being fixed. While reading a component from stream, it can happen that the stream contains a component reference property with a name of a component that was not yet created and read from the stream. Such properties are saved, and the missing references are resolved when the complete stream was read. This resolving step is called fixing up references, and the `csFixups` flags is set during this step.

csFreeNotification This flag indicates that the component has free notifications registered with TComponent.FreeNotification (379)

```
TComponentStyle= Set of (csInheritable,csCheckPropAvail,csSubComponent
  ,
                         csTransient)
```

Table 4.7: Enumeration values for type

Value	Explanation
csCheckPropAvail	??
csInheritable	The component can be on inherited forms.
csSubComponent	Subcomponent - streamed as part of the owning component.
csTransient	Transient component.

Describes the style of the component.

```
TCreateComponentEvent = procedure(Reader: TReader;
  ComponentClass: TComponentClass;
  var Component: TComponent) of object
```

Event handler type, occurs when a component instance must be created when a component is read from a stream.

```
TDataModuleClass = Class of TDataModule
```

`TDataModuleClass` defines the class pointer for TDataModule (389).

```
TDuplicates = Types.TDuplicates
```

Type to describe what to do with duplicate values in a TStringlist (470).

```
TExceptionClass = Class of Exception
```

`TExceptionClass` is the class pointer for the Exception (1839) class, defined in the SysUtils (1635) unit.

```
TFilerFlag = (ffInherited,ffChildPos,ffInline)
```

Table 4.8: Enumeration values for type TFilerFlag

Value	Explanation
ffChildPos	The position of the child on it's parent is included.
ffInherited	Stored object is an inherited object.
ffInline	Used for frames.

The `TFiler` class uses this enumeration type to decide whether the streamed object was streamed as part of an inherited form or not.

```
TFilerFlags = Set of TFilerFlag
```

Set of TFilerFlag (279).

```
TFindAncestorEvent = procedure(Writer: TWriter; Component: TComponent
  ;
                                const Name: string;
  var Ancestor: TComponent;
  var RootAncestor: TComponent) of object
```

Event that occurs w.

```
TFindComponentClassEvent = procedure(Reader: TReader;
  const ClassName: string;
  var ComponentClass: TComponentClass)
  of object
```

Event handler type, occurs when a component class pointer must be found when reading a component from a stream.

```
TFindGlobalComponent = function(const Name: string) : TComponent
```

`TFindGlobalComponent` is a callback used to find a component in a global scope. It is used when the streaming system needs to find a component which is not part of the component which is currently being streamed. It should return the component with name `Name`, or `Nil` if none is found.

The variable FindGlobalComponent (297) is a callback of type `TFindGlobalComponent`. It can be set by the IDE when an unknown reference is found, to offer the designer to redirect the link to a new component.

```
TFindMethodEvent = procedure(Reader: TReader; const MethodName: string
  ;
                             var Address: CodePointer;
  var Error: Boolean) of object
```

If a TReader (440) instance needs to locate a method and it doesn't find it in the streamed form, then the OnFindMethod (451) event handler will be called, if one is installed. This event can be assigned in order to use different locating methods. If a method is found, then its address should be returned in `Address`. The `Error` should be set to `True` if the reader should raise an exception after the event was handled. If it is set to `False` no exception will be raised, even if no method was found. On entry, `Error` will be set to `True`.

```
TFPObservedOperation = (ooChange,ooFree,ooAddItem,ooDeleteItem,ooCustom
  )
```

Table 4.9: Enumeration values for type TFPObservedOperation

Value	Explanation
ooAddItem	An item is added to the observed object (generally a list).
ooChange	The observed object has changed.
ooCustom	Custom event.
ooDeleteItem	An item is deleted from the observed object (generally a list).
ooFree	The observed object is being freed.

`TFPObservedOperation` enumerates the possible operations that can be reported to an observer. Which of these operations is reported depends on the implementation of the observed object.

```
TGetChildProc = procedure(Child: TComponent) of object
```

Callback used when obtaining child components.

```
TGetStrProc = procedure(const S: string) of object
```

This event is used as a callback to retrieve string values. It is used, among other things, to pass along string properties in property editors.

```
THandle = System.THandle
```

This type is used as the handle for THandleStream (406) stream descendants

```
THelpContext = - MaxLongint..MaxLongint
```

Range type to specify help contexts.

```
THelpEvent = function(Command: Word; Data: LongInt;
  var CallHelp: Boolean) : Boolean of object
```

This event is used for display of online help.

```
THelpType = (htKeyword,htContext)
```

Table 4.10: Enumeration values for type THelpType

Value	Explanation
htContext	Help type: Context ID help.
htKeyword	Help type: Keyword help.

Enumeration type specifying the kind of help requested.

```
THintEvent = procedure(var HintStr: string; var CanShow: Boolean)
  of object
```

```
TIdentToInt = function(const Ident: string; var Int: LongInt) :
  Boolean
```

`TIdentToInt` is a callback used to look up identifiers (`Ident`) and return an integer value corresponding to this identifier (`Int`). The callback should return `True` if a value corresponding to integer `Ident` was found, `False` if not.

A callback of type `TIdentToInt` should be specified when an integer is registered using the RegisterIntegerConsts (306) call.

```
TInitComponentHandler = function(Instance: TComponent;
  RootAncestor: TClass) : Boolean
```

`TInitComponentHandler` is a callback type. It is used in the InitInheritedComponent (300) call to initialize a component. Callbacks of this type are registered with the RegisterInitComponentHandler (305) call.

```
TIntToIdent = function(Int: LongInt; var Ident: string) : Boolean
```

`TIdentToInt` is a callback used to look up integers (`Ident`) and return an identifier (`Ident`) that can be used to represent this integer value in an IDE. The callback should return `True` if a value corresponding to integer `Ident` was found, `False` if not.

A callback of type `TIntToIdent` should be specified when an integer is registered using the RegisterIntegerConsts (306) call.

```
TLeftRight = taLeftJustify..taRightJustify
```

`TLeftRight` is a subrange type based on the TAlignment (276) enumerated type. It contains only the left and right alignment constants.

```
TListAssignOp = (laCopy,laAnd,laOr,laXor,laSrcUnique,laDestUnique
  )
```

Table 4.11: Enumeration values for type TListAssignOp

Value	Explanation
laAnd	Remove all elements not first second list.
laCopy	Clear list and copy all strings from second list.
laDestUnique	Keep all elements that exists only in list2.
laOr	Add all elements from second (and optional third) list, eliminate duplicates.
laSrcUnique	Just keep all elements that exist only in source list.
laXor	Remove elements in second lists, Add all elements from second list not in first list.

This type determines what operation TList.Assign (421) or TFPList.assign (402) performs.

```
TListCallback = Types.TListCallback
```

`TListCallback` is the method callback prototype for the function that is passed to the TFPList.ForEachCall (403) call. The `data` argument will be filled with all the pointers in the list (one per call) and the `arg` argument is the `Arg` argument passed to the `ForEachCall` call.

```
TListNotification = (lnAdded,lnExtracted,lnDeleted)
```

Table 4.12: Enumeration values for type TListNotification

Value	Explanation
lnAdded	List change notification: Element added to the list.
lnDeleted	List change notification: Element deleted from the list.
lnExtracted	List change notification: Element extracted from the list.

Kind of list notification event.

```
TListSortCompare = function(Item1: Pointer; Item2: Pointer) :
  Integer
```

Callback type for the list sort algorithm.

```
TListStaticCallback = Types.TListStaticCallback
```

TListCallback is the procedurald callback prototype for the function that is passed to the TFPList.ForEachCall (403) call. The data argument will be filled with all the pointers in the list (one per call) and the arg argument is the Arg argument passed to the ForEachCall call.

```
TMissingNameValueSeparatorAction = (mnvaValue,mnvaName,mnvaEmpty,
  mnvaError)
```

Table 4.13: Enumeration values for type TMissingNameValueSeparatorAction

Value	Explanation
mnvaEmpty	Consider it an empty line.
mnvaError	Raise an exception if the Name=Value functionality is used.
mnvaName	Consider the line a name with empty value.
mnvaValue	Consider the line a value.

TMissingNameValueSeparatorAction is the type of the TStrings.MissingNameValueSeparatorAction (495) property and determines what to do if in a TStrings (475) descendent the NameValueSeparator (496) is missing. It can have the following values:

mnvaValue Consider the line a value.

mnvaName Consider the line a name with empty value.

mnvaEmpty Consider it an empty line.

mnvaError Raise an exception if the Name=Value functionality is used.

```
TMissingNameValueSeparatorActions = Set of
  TMissingNameValueSeparatorAction
```

TMissingNameValueSeparatorActions is simply a set of TMissingNameValueSeparatorAction (283) values.

```
TNotifyCallBack = procedure(Sender: TObject; AData: Pointer)
```

TNotifyCallBack is used to notify about thread termination when using static callbacks. When called, the Sender will contain the thread which is terminating, and AData is the AData parameter passed to TThread.ExecuteInThread (512).

```
TNotifyEvent = procedure(Sender: TObject) of object
```

Most event handlers are implemented as a property of type TNotifyEvent. When this is set to a certain method of a class, when the event occurs, the method will be called, and the class that generated the event will pass itself along as the Sender argument.

```
TObjectTextEncoding = (oteDFM,oteLFM)
```

Table 4.14: Enumeration values for type TObjectTextEncoding

Value	Explanation
oteDFM	Characters are in DFM (Delphi) format: widechar encoded.
oteLFM	Characters are in LFM format: UTF-8 encoded.

`TObjectTextEncoding` is an enumerated type which denotes the encoding of non ASCII characters in an object stream file. It is needed for correct encoding when reading string values in the text stream.

```
TOperation = (opInsert,opRemove)
```

Table 4.15: Enumeration values for type TOperation

Value	Explanation
opInsert	A new component is being inserted in the child component list.
opRemove	A component is being removed from the child component list.

Operation of which a component is notified.

```
TPersistentClass = Class of TPersistent
```

`TPersistentClass` is the class reference type for the TPersistent (435) class.

```
TPoint = Types.TPoint
```

This record describes a coordinate. It is used to handle the Top (376) and Left (376) properties of TComponent (376).

`X` represents the X-Coordinate of the point described by the record. `Y` represents the Y-Coordinate of the point described by the record.

```
TPointerList = Array[0..MaxListSize-1] of Pointer
```

Type for an Array of pointers.

```
TPropertyNotFoundEvent = procedure(Reader: TReader;
  Instance: TPersistent;
  var PropName: string;
  IsPath: Boolean;
                                    var
  Handled: Boolean;
                                    var Skip: Boolean
  ) of object
```

TPropertyNotFoundEvent is the prototype for the TReader.OnPropertyNotFound (450) event. Reader is the sender of the event, Instance is the instance that is being streamed. PropInfo is a pointer to the RTTI information for the property being read. Handled should be set to True if the handler redirected the unknown property successfully, and Skip should be set to True if the value should be skipped. IsPath determines whether the property refers to a sub-property.

```
TReadComponentsProc = procedure(Component: TComponent) of object
```

Callback type when reading a component from a stream.

```
TReaderError = procedure(Reader: TReader; const Message: string;
  var Handled: Boolean) of object
```

Event handler type, called when an error occurs during the streaming.

```
TReaderProc = procedure(Reader: TReader) of object
```

The TReaderProc reader procedure is a callback procedure which will be used by a TPersistent (435) descendant to read user properties from a stream during the streaming process. The Reader argument is the writer object which can be used read properties from the stream.

```
TReadWriteStringPropertyEvent = procedure(Sender: TObject;
  const Instance: TPersistent;
  PropInfo: PPropInfo;
  var Content: string) of
  object
```

TReadWriteStringPropertyEvent is the prototype for the TReader.OnReadStringProperty (452) event handler. Reader is the sender of the event, Instance is the instance that is being streamed. PropInfo is a pointer to the RTTI information for the property being read. Content is the string as it was read from the stream.

```
TRect = Types.TRect
```

TRect describes a rectangle in space with its upper-left (in (Top,Left>)) and lower-right (in (Bottom ,Right)) corners.

```
TReferenceNameEvent = procedure(Reader: TReader; var Name: string
  )
                                    of object
```

Occurs when a named object needs to be looked up.

```
TSeekOrigin = (soBeginning,soCurrent,soEnd)
```

Table 4.16: Enumeration values for type TSeekOrigin

Value	Explanation
soBeginning	Offset is interpreted relative to the start of the stream.
soCurrent	Offset is interpreted relative to the current position in the stream.
soEnd	Offset is interpreted relative to the end of the stream.

Specifies the origin of the TStream.Seek (457) method.

```
TSetMethodPropertyEvent = procedure(Reader: TReader;
  Instance: TPersistent;
  PropInfo: PPropInfo;
  const TheMethodName: string;
  var Handled: Boolean) of object
```

`TSetMethodPropertyEvent` is the prototype for the TReader.OnSetMethodProperty (451) event. `Reader` is the sender of the event, `Instance` is the instance that is being streamed. `PropInfo` is a pointer to the RTTI information for the property being read, and `TheMethodName` is the name of the method that the property should be set to. `Handled` should be set to `True` if the handler set the property successfully.

```
TSetNameEvent = procedure(Reader: TReader; Component: TComponent;
  var Name: string) of object
```

Occurs when the reader needs to set a component's name.

```
TShiftState = Set of TShiftStateEnum
```

This type is used when describing a shortcut key or when describing what special keys are pressed on a keyboard when a key event is generated.

The set contains the special keys that can be used in combination with a 'normal' key.

```
TShiftStateEnum = (ssShift,ssAlt,ssCtrl,ssLeft,ssRight,ssMiddle,
  ssDouble,ssMeta,ssSuper,ssHyper,ssAltGr,ssCaps
  ,ssNum,
                   ssScroll,ssTriple,ssQuad,ssExtra1,ssExtra2
  )
```

Table 4.17: Enumeration values for type TShiftStateEnum

Value	Explanation
ssAlt	Alt key pressed.
ssAltGr	Alt-GR key pressed.
ssCaps	Caps lock key pressed.
ssCtrl	Ctrl key pressed.
ssDouble	Double mouse click.
ssExtra1	Extra key 1.
ssExtra2	Extra key 2.
ssHyper	Hyper key pressed.
ssLeft	Left mouse button pressed.
ssMeta	Meta key pressed.
ssMiddle	Middle mouse button pressed.
ssNum	Num lock key pressed.
ssQuad	Quadruple mouse click.
ssRight	Right mouse button pressed.
ssScroll	Scroll lock key pressed.
ssShift	Shift key pressed.
ssSuper	Super key pressed.
ssTriple	Triple mouse click.

Keyboard/Mouse shift state enumerator.

```
TShortCut = (Word)..(Word)
```

Enumeration type to identify shortcut key combinations.

```
TSmallPoint = Types.TSmallPoint
```

Same as TPoint (284), only the X and Y ranges are limited to 2-byte integers instead of 4-byte integers.

```
TStreamOwnership = (soReference,soOwned)
```

Table 4.18: Enumeration values for type TStreamOwnership

Value	Explanation
soOwned	Stream is owned: it will be freed when the adapter is freed.
soReference	Stream is referenced only, it is not freed by the adapter.

The ownership of a streamadapter determines what happens with the stream on which a TStreamAdapter (465) acts, when the adapter is freed.

```
TStreamProc = procedure(Stream: TStream) of object
```

Procedure type used in streaming.

```
TStringItemList = Array[0..MaxListSize] of TStringItem
```

This declaration is provided for Delphi compatibility, it is not used in Free Pascal.

```
TStringListSortCompare = function(List: TStringList; Index1: Integer
  ;
                                  Index2: Integer) : Integer
```

Callback type used in stringlist compares.

```
TStringsClass = Class of TStrings
```

`TStringsClass` is the class of TStrings (475). It is provided for convenience.

```
TStringsFilterMethod = function(const s: string) : Boolean of
  object
```

`TStringsFilterMethod` is the prototype for the callback in TStrings.Filter (485). Return `True` if the string `S` must be included in the TStrings.Filter (485) result, `False` if it must be excluded.

```
TStringsForEachMethod = procedure(const CurrentValue: string) of
  object
```

`TStringsForEachMethod` is a callback signature for TStrings.Foreach (485). It gets passed the string value in `CurrentValue`.

```
TStringsForEachMethodEx = procedure(const CurrentValue: string;
  const index: Integer) of
  object
```

TStringsForEachMethodEx is a callback signature for TStrings.Foreach (485). It gets passed the string value in CurrentValue, string index in Index.

```
TStringsForEachMethodExObj = procedure(const CurrentValue: string
  ;
                                       const index: Integer;
  Obj: TObject) of object
```

TStringsForEachMethodExObj is a callback signature for TStrings.Foreach (485). It gets passed the string value in CurrentValue, string index in Index and associated object in Obj.

```
TStringsMapMethod = function(const s: string) : string of object
```

TStringsMapMethod is the prototype for the callback in TStrings.Map (489). The method must return the mapped value for value S.

```
TStringsOption = (soStrictDelimiter,soWriteBOM,soTrailingLineBreak
  ,
                  soUseLocale,soPreserveBOM)
```

Table 4.19: Enumeration values for type TStringsOption

Value	Explanation
soPreserveBOM	If this option is enabled, TStrings.WriteBOM (500) is set in TStrings.LoadFromStream (489) or TStr
soStrictDelimiter	See TStrings.StrictDelimiter (498).
soTrailingLineBreak	See TStrings.TrailingLineBreak (497).
soUseLocale	See TStrings.UseLocale (499).
soWriteBOM	See TStrings.WriteBOM (500).

TStringsOption is the type used in the TStrings.Options (497) set property, and it lists various strings options. It has the following values:

soStrictDelimiter See TStrings.StrictDelimiter (498).

soWriteBOM See TStrings.WriteBOM (500).

soTrailingLineBreak See TStrings.TrailingLineBreak (497).

soUseLocale See TStrings.UseLocale (499).

soPreserveBOM If this option is enabled, TStrings.WriteBOM (500) is set in TStrings.LoadFromStream (489) or TStrings.LoadFromFile (488) according to BOM presence in the loaded file.

```
TStringsOptions = Set of TStringsOption
```

TStringsOptions is the set type for enumeration TStringsOption (288).

```
TStringsReduceMethod = function(const s1: string; const s2: string
  )
                                    : string of object
```

`TStringsReduceMethod` is the prototype for the callback in TStrings.Reduce (490). The method must return the reduced value for values `S1` and `S2`. On every iteration `S1` will contain the result of the previous iteration.

```
TStringsSortStyle = (sslNone,sslUser,sslAuto)
```

Table 4.20: Enumeration values for type TStringsSortStyle

Value	Explanation
sslAuto	The `TStrings` instance keeps the strings sorted.
sslNone	The strings are not sorted.
sslUser	The strings are kept sorted by the user.

`TStringsSortStyle` is the type used in the TStringList.SortStyle (475) property, and describes how the strings are sorted when the TStringList.Sorted (474) property is `True`. It has the following values:

sslNone The strings are not sorted.

sslUser The strings are kept sorted by the user.

sslAuto The `TStrings` instance keeps the strings sorted.

```
TStringsSortStyles = Set of TStringsSortStyle
```

`TStringsSortStyles` is the set type for enumeration TStringsSortStyle (289).

```
TSynchronizeProcVar = procedure
```

Synchronize callback type.

```
TThreadExecuteCallBack = procedure(AData: Pointer)
```

`TThreadExecuteCallBack` is the signature of the static procedure to be used when executing something in a thread using TThread.ExecuteInThread (512) when no status reporting is required.

```
TThreadExecuteHandler = TThreadMethod
```

`TThreadExecuteHandler` is the signature of the method to be used when executing something in a thread using TThread.ExecuteInThread (512) when no status reporting is required.

```
TThreadExecuteStatusCallBack = procedure(AData: Pointer;
  ReportStatus: TThreadReportStatus
  )
```

TThreadExecuteStatusCallBack is the signature of a procedure to be used when executing something in a thread using TThread.ExecuteInThread (512) when status reporting is required.

On entry in the method, AData is the AData parameter passed to TThread.ExecuteInThread (512). ReportStatus is passed to the method, and the threaded procedure can call ReportStatus at various stages to report about the status of the method: The status will be reported to the main thread using TThread.synchronize (508), so calls to ReportStatus will be blocked as long as the status was not reported.

```
TThreadExecuteStatusHandler = procedure
  (ReportStatus: TThreadReportStatus)
  of object
```

TThreadExecuteStatusHandler is the signature of the method to be used when executing something in a thread using TThread.ExecuteInThread (512) when status reporting is required.

On entry in the method, ReportStatus is passed to the method, and the method can call ReportStatus at various stages to report about the status of the method: The status will be reported to the main thread using TThread.synchronize (508), and calls ReportStatus will be blocked as long as the status was not reported.

```
TThreadMethod = procedure of object
```

Procedure variable used when synchronizing threads.

```
TThreadPriority = (tpIdle,tpLowest,tpLower,tpNormal,tpHigher,tpHighest
  ,
                   tpTimeCritical)
```

Table 4.21: Enumeration values for type TThreadPriority

Value	Explanation
tpHigher	Thread runs at high priority.
tpHighest	Thread runs at highest possible priority.
tpIdle	Thread only runs when other processes are idle.
tpLower	Thread runs at a lower priority.
tpLowest	Thread runs at the lowest priority.
tpNormal	Thread runs at normal process priority.
tpTimeCritical	Thread runs at realtime priority.

Enumeration specifying the priority at which a thread runs.

```
TThreadReportStatus = procedure(const status: string) of object
```

TThreadReportStatus is the callback prototype for the TThread.ExecuteInThread (512) method. This callback is used to report thread status to the main thread: the Status string can be used to report the status of thread execution.

```
TThreadStatusNotifyCallBack = procedure(Sender: TThread;
  AData: Pointer;
  const status: string)
```

TThreadStatusNotifyCallBack is the signature of the callback to be provided when executing a static procedure in a thread using TThread.ExecuteInThread (512) when status reporting is required.

This callback will be called in the main thread. When called, it has the thread whose status is reported in Sender, AData is the AData parameter passed to TThread.ExecuteInThread (512). Finally, the status message in ReportStatus is passed to the callback.

Note that the thread reporting its status is blocked while the callback is being handled: The status is be reported to the main thread using TThread.synchronize (508).

```
TThreadStatusNotifyEvent = procedure(Sender: TThread;
  const status: string) of object
```

TThreadStatusNotifyEvent is the signature of the method to be provided when executing something in a thread using TThread.ExecuteInThread (512) when status reporting is required.

This event handler will be called in the main thread. When called, it has the thread whose status is reported in Sender, and the status message in ReportStatus is passed to the method.

Note that the thread reporting its status is blocked while the callback is being handled: The status is be reported to the main thread using TThread.synchronize (508).

```
TTopBottom = taAlignTop..taAlignBottom
```

TTopBottom is a subrange of TVerticalAlignment (292) that excludes the vertically centered alignment value.

```
TValueType = (vaNull,vaList,vaInt8,vaInt16,vaInt32,vaExtended,vaString
  ,
              vaIdent,vaFalse,vaTrue,vaBinary,vaSet,vaLString,vaNil
  ,
              vaCollection,vaSingle,vaCurrency,vaDate,vaWString
  ,vaInt64,
              vaUTF8String,vaUString,vaQWord)
```

Table 4.22: Enumeration values for type TValueType

Value	Explanation
vaBinary	Binary data follows.
vaCollection	Collection follows.
vaCurrency	Currency value follows.
vaDate	Date value follows.
vaExtended	Extended value.
vaFalse	Boolean `False` value.
vaIdent	Identifier.
vaInt16	Integer value, 16 bits long.
vaInt32	Integer value, 32 bits long.
vaInt64	Integer value, 64 bits long.
vaInt8	Integer value, 8 bits long.
vaList	Identifies the start of a list of values.
vaLString	Ansistring data follows.
vaNil	Nil pointer.
vaNull	Empty value. Ends a list.
vaQWord	QWord (64-bit word) value.
vaSet	Set data follows.
vaSingle	Single type follows.
vaString	String value.
vaTrue	Boolean `True` value.
vaUString	UnicodeString value.
vaUTF8String	UTF8 encoded Unicode string.
vaWString	Widestring value follows.

Enumerated type used to identify the kind of streamed property.

```
TVerticalAlignment = (taAlignTop,taAlignBottom,taVerticalCenter)
```

Table 4.23: Enumeration values for type TVerticalAlignment

Value	Explanation
taAlignBottom	Align vertically to the bottom.
taAlignTop	Align vertically to the top.
taVerticalCenter	Align vertically centered.

`TVerticalAlignment` can be used to specity a vertical alignment. It is not used in the classes unit, and is provided for symmetry with TAlignment (276). It can have the following values:

taAlignTop Align vertically to the top.

taAlignBottom Align vertically to the bottom.

taVerticalCenter Align vertically centered.

```
TWriteMethodPropertyEvent = procedure(Writer: TWriter;
  Instance: TPersistent;
  PropInfo: PPropInfo;
```

```
  const MethodValue: TMethod;
  const DefMethodValue: TMethod;
  var Handled: Boolean) of object
```

`TWriteMethodPropertyEvent` is the prototype for the TWriter.OnWriteMethodProperty (531) event. `Writer` is the sender of the event, `Instance` is the instance that is being streamed. `PropInfo` is a pointer to the RTTI information for the property being written, and `MethodValue` is the value of the method that the property was set to. `DefMethodCodeValue` is set to the default value of the property (Nil or the parent value). `Handled` should be set to `True` if the handler set the property successfully.

```
TWriterProc = procedure(Writer: TWriter) of object
```

The `TWriterProc` writer procedure is a callback procedure which will be used by a TPersistent (435) descendant to write user properties from a stream during the streaming process. The `Writer` argument is the writer object which can be used write properties to the stream.

4.3.3 Variables

```
AddDataModule : procedure(DataModule: TDataModule) of object
```

`AddDataModule` can be set by an IDE or a streaming mechanism to receive notification when a new instance of a TDataModule (389) descendant is created.

```
ApplicationHandleException : procedure(Sender: TObject) of object
```

`ApplicationHandleException` can be set by an application object to handle any exceptions that may occur when a TDataModule (389) is created.

```
ApplicationShowException : procedure(E: Exception) of object
```

Unused.

```
CreateVCLComObjectProc : procedure(Component: TComponent) = Nil
```

`CreateVCLComObjectProc` is called by `TComponent` if it needs to create a `IVCLComObject` interface for itself (when the `ComObject` property is read). It passes itself as the `Component` parameter.

```
GlobalNameSpace : IReadWriteSync
```

An interface protecting the global namespace. Used when reading/writing to the global namespace list during streaming of forms.

```
MainThreadID : TThreadID
```

ID of main thread. Unused at this point.

```
RegisterComponentsProc : procedure(const Page: string;
  ComponentClasses: Array of TComponentClass)
```

RegisterComponentsProc can be set by an IDE to be notified when new components are being registered. Application programmers should never have to set RegisterComponentsProc

```
RegisterNoIconProc : procedure(ComponentClasses: Array of TComponentClass
  )
```

RegisterNoIconProc can be set by an IDE to be notified when new components are being registered, and which do not need an Icon in the component palette. Application programmers should never have to set RegisterComponentsProc

```
RemoveDataModule : procedure(DataModule: TDataModule) of object
```

RemoveDataModule can be set by an IDE or a streaming mechanism to receive notification when an instance of a TDataModule (389) descendant is freed.

```
WakeMainThread : TNotifyEvent = Nil
```

WakeMainThread is a handler, which, when set, is called by the TThread.Synchronize (508) routine to signal the main thread that a synchronization routine is waiting in the queue.

This handler is by default empty. An actual implementation depends on the main program logic (usually an event loop) and must be provided by the event loop logic: the event loop will normally call CheckSynchronize (295) at regular intervals. The WakeMainThread can make sure this happens as soon as possible.

While this handle should alert the main program thread that a thread is waiting for synchronization, the call is executed by the thread, and should therefore NOT synchronize the thread, but should somehow signal the main thread that a thread is waiting for synchronization. For example, by sending a message.

4.4 Procedures and functions

4.4.1 ActivateClassGroup

Synopsis: Activates a class group.

Declaration: `function ActivateClassGroup(AClass: TPersistentClass) : TPersistentClass`

Visibility: default

Description: ActivateClassGroup activates the group of classes to which AClass belongs. The function returns the class that was last used to activate the class group.

The class registration and streaming mechanism allows to organize the classes in groups. This allows an IDE to form groups of classes, which can be enabled or disabled. It is not needed at Run-Time.

Errors: If AClass does not belong to a class group, an exception is raised.

See also: StartClassGroup (307), GroupDescendentsWith (299), ClassGroupOf (296)

4.4.2 BeginGlobalLoading

Synopsis: Not yet implemented.

Declaration: `procedure BeginGlobalLoading`

Visibility: default

Description: Not yet implemented.

4.4.3 BinToHex

Synopsis: Convert a binary buffer to a hexadecimal string.

Declaration:
```
procedure BinToHex(BinValue: PChar; HexValue: PChar;
                   BinBufSize: Integer)
```

Visibility: default

Description: `BinToHex` converts the byte values in `BinValue` to a string consisting of 2-charachter hexadecimal strings in `HexValue`. `BufSize` specifies the length of `BinValue`, which means that `HexValue` must have size `2*BufSize`.

For example a buffer containing the byte values 255 and 0 will be converted to FF00.

Errors: No length checking is done, so if an invalid size is specified, an exception may follow.

See also: HexToBin (299)

4.4.4 Bounds

Synopsis: Returns a `TRect` structure with the bounding rect of the given location and size.

Declaration:
```
function Bounds(ALeft: Integer; ATop: Integer; AWidth: Integer;
                AHeight: Integer) : TRect
```

Visibility: default

Description: `Bounds` returns a TRect (285) record with the given origin `(ALeft,ATop)` and dimensions `(AWidth,AHeight)` filled in. The bottom-right corner is calculated by adding `AWidth` to `ALeft` and `AHeight` to `ATop`. As a result, a rectangle with width/height set to 0 is exactly 1 pixel.

See also: Rect (304)

4.4.5 CheckSynchronize

Synopsis: Check whether there are any synchronize calls in the synchronize queue.

Declaration:
```
function CheckSynchronize(timeout: LongInt) : Boolean
```

Visibility: default

Description: `CheckSynchronize` should be called regularly by the main application thread to handle any TThread.Synchronize (508) calls that may be waiting for execution by the main thread. If any such calls are waiting for execution by the main thread, they are executed at once, in the order that they were scheduled.

The function returns `True` if any `Synchronize` method was executed.

`TimeOut` is the maximum amount of time (in milliseconds) that the `CheckSynchronize` routine will wait for synchronisation requests to appear in the queue.

Calling this routine more often will ensure that synchronize requests are handled faster.

This routine may not be called from any thread other than the main thread, as it will execute the waiting requests.

Threads may call the WakeMainThread (294) to signal the main thread that the synchronisation queue contains items, and thus speed up the execution of the synchronize calls.

See also: TThread.Synchronize (508), WakeMainThread (294)

4.4.6 ClassGroupOf

Synopsis: Returns the class group to which an instance or class belongs.

Declaration:
```
function ClassGroupOf(AClass: TPersistentClass) : TPersistentClass
function ClassGroupOf(Instance: TPersistent) : TPersistentClass
```

Visibility: default

Description: `ClassGroupOf` returns the class group to which `AClass` or `Instance` belongs.

Errors: The result is `Nil` if no matching class group is found.

See also: StartClassGroup (307), ActivateClassGroup (294), GroupDescendentsWith (299)

4.4.7 CollectionsEqual

Synopsis: Returns `True` if two collections are equal.

Declaration:
```
function CollectionsEqual(C1: TCollection; C2: TCollection) : Boolean
function CollectionsEqual(C1: TCollection; C2: TCollection;
                          Owner1: TComponent; Owner2: TComponent)
                           : Boolean
```

Visibility: default

Description: `CollectionsEqual` is not yet implemented. It simply returns `False`

4.4.8 EndGlobalLoading

Synopsis: Not yet implemented.

Declaration:
```
procedure EndGlobalLoading
```

Visibility: default

Description: Not yet implemented.

4.4.9 ExtractStrings

Synopsis: Split a string in different words.

Declaration:
```
function ExtractStrings(Separators: TSysCharSet;
                        WhiteSpace: TSysCharSet; Content: PChar;
                        Strings: TStrings; AddEmptyStrings: Boolean)
                         : Integer
```

Visibility: default

Description: `ExtractStrings` splits `Content` (a null-terminated string) into words, and adds the words to the `Strings` stringlist. The words are separated by `Separators` and any characters in `whitespace` are stripped from the strings. The space and CR/LF characters are always considered whitespace.

Errors: No length checking is performed on `Content`. If no null-termination character is present, an access violation may occur. Likewise, if `Strings` is not valid, an access violation may occur.

4.4.10 FindClass

Synopsis: Returns the class pointer of a class with given name.

Declaration: `function FindClass(const AClassName: string) : TPersistentClass`

Visibility: default

Description: `FindClass` searches for the class named `ClassName` in the list of registered classes and returns a class pointer to the definition. If no class with the given name could be found, an exception is raised.

The GetClass (298) function does not raise an exception when it does not find the class, but returns a `Nil` pointer instead.

See also: RegisterClass (304), GetClass (298)

4.4.11 FindGlobalComponent

Synopsis: Callback used when a component must be found.

Declaration: `function FindGlobalComponent(const Name: string) : TComponent`

Visibility: default

Description: `FindGlobalComponent` is a callback of type TFindGlobalComponent (280). It can be set by the IDE when an unknown reference is found, to offer the user to redirect the link to a new component.

It is a callback used to find a component in a global scope. It is used when the streaming system needs to find a component which is not part of the component which is currently being streamed. It should return the component with name `Name`, or `Nil` if none is found.

See also: TFindGlobalComponent (280)

4.4.12 FindIdentToInt

Synopsis: Return the string to integer converter for an integer type.

Declaration: `function FindIdentToInt(AIntegerType: Pointer) : TIdentToInt`

Visibility: default

Description: `FindIdentToInt` returns the handler that handles the conversion of a string representation to an integer that can be used in component streaming, when IdentToInt (299) is called.

Errors: Nil is returned if no handler is registered for the given type.

4.4.13 FindIntToIdent

Synopsis: Return the integer to string converter for an integer type.

Declaration: `function FindIntToIdent(AIntegerType: Pointer) : TIntToIdent`

Visibility: default

Description: `FindIntToIdent` returns the handler that handles the conversion of an integer to a string representation that can be used in component streaming, when IntToIdent (300) is called.

Errors: Nil is returned if no handler is registered for the given type.

See also: IntToIdent (300), TIntToIdent (282), FindIdentToInt (297)

4.4.14 FindNestedComponent

Synopsis: Finds the component with name path starting at the indicated root component.

Declaration:
```
function FindNestedComponent(Root: TComponent; APath: string;
                             CStyle: Boolean) : TComponent
```

Visibility: default

Description: `FindNestedComponent` will descend through the list of owned components (starting at `Root`) and will return the component whose name path matches `NamePath`. As a path separator the characters . (dot), - (dash) and > (greater than) can be used

See also: GlobalFixupReferences (299)

4.4.15 GetClass

Synopsis: Returns the class pointer of a class with given name.

Declaration:
```
function GetClass(const AClassName: string) : TPersistentClass
```

Visibility: default

Description: `GetClass` searches for the class named `ClassName` in the list of registered classes and returns a class pointer to the definition. If no class with the given name could be found, `Nil` is returned.

The FindClass (297) function will raise an exception if the does not find the class.

See also: RegisterClass (304), GetClass (298)

4.4.16 GetFixupInstanceNames

Synopsis: Returns the names of elements that need to be resolved for the `root` component, whose reference contains `ReferenceRootName`.

Declaration:
```
procedure GetFixupInstanceNames(Root: TComponent;
                                const ReferenceRootName: string;
                                Names: TStrings)
```

Visibility: default

Description: `GetFixupInstanceNames` examines the list of unresolved references and returns the names of classes that contain unresolved references to the `Root` component in the list `Names`. The list is not cleared prior to filling it.

See also: GetFixupReferenceNames (298), GlobalFixupReferences (299)

4.4.17 GetFixupReferenceNames

Synopsis: Returns the names of elements that need to be resolved for the `root` component.

Declaration:
```
procedure GetFixupReferenceNames(Root: TComponent; Names: TStrings)
```

Visibility: default

Description: `GetFixupReferenceNames` examines the list of unresolved references and returns the names of properties that must be resolved for the component `Root` in the list `Names`. The list is not cleared prior to filling it.

See also: GetFixupInstanceNames (298), GlobalFixupReferences (299)

4.4.18 GlobalFixupReferences

Synopsis: Called to resolve unresolved references after forms are loaded.

Declaration:
```
procedure GlobalFixupReferences
```

Visibility: default

Description: `GlobalFixupReferences` runs over the list of unresolved references and tries to resolve them. This routine should under normal circumstances not be called in an application programmer's code. It is called automatically by the streaming system after a component has been instantiated and its properties read from a stream. It will attempt to resolve references to other global components.

See also: GetFixupReferenceNames (298), GetFixupInstanceNames (298)

4.4.19 GroupDescendentsWith

Synopsis: Provided for Delphi compatibility.

Declaration:
```
procedure GroupDescendentsWith(AClass: TPersistentClass;
                               AClassGroup: TPersistentClass)
```

Visibility: default

Description: `GroupDescendentsWith` exists for Delphi compatibility, it doesn't actually do anything in Free Pascal.

See also: RegisterClasses (305)

4.4.20 HexToBin

Synopsis: Convert a hexadecimal string to a binary buffer.

Declaration:
```
function HexToBin(HexValue: PChar; BinValue: PChar; BinBufSize: Integer)
                  : Integer
```

Visibility: default

Description: `HexToBin` scans the hexadecimal string representation in `HexValue` and transforms every 2 character hexadecimal number to a byte and stores it in `BinValue`. The buffer size is the size of the binary buffer. Scanning will stop if the size of the binary buffer is reached or when an invalid character is encountered. The return value is the number of stored bytes.

Errors: No length checking is done, so if an invalid size is specified, an exception may follow.

See also: BinToHex (295)

4.4.21 IdentToInt

Synopsis: Looks up an integer value in a integer-to-identifier map list.

Declaration:
```
function IdentToInt(const Ident: string; out Int: LongInt;
                    const Map: Array of TIdentMapEntry) : Boolean
```

Visibility: default

Description: `IdentToInt` searches `Map` for an entry whose `Name` field matches `Ident` and returns the corresponding integer value in `Int`. If a match was found, the function returns `True`, otherwise, `False` is returned.

See also: TIdentToInt (281), TIntToIdent (282), IntToIdent (300), TIdentMapEntry (309)

4.4.22 IfThen

Declaration: `function IfThen(AValue: Boolean; const ATrue: TStringList; const AFalse: TStringList) : TStringList; Overload`

Visibility: default

4.4.23 InitComponentRes

Synopsis: Provided for Delphi compatibility only.

Declaration: `function InitComponentRes(const ResName: string; Instance: TComponent) : Boolean`

Visibility: default

Description: This function is provided for Delphi compatibility. It always returns `false`.

See also: ReadComponentRes (303)

4.4.24 InitInheritedComponent

Synopsis: Initializes a component descending from `RootAncestor`.

Declaration: `function InitInheritedComponent(Instance: TComponent; RootAncestor: TClass) : Boolean`

Visibility: default

Description: `InitInheritedComponent` should be called from a constructor to read properties of the component `Instance` from the streaming system. The `RootAncestor` class is the root class from which `Instance` is a descendant. This must be one of `TDatamodule`, `TCustomForm` or `TFrame`.

The function returns `True` if the properties were successfully read from a stream or `False` if some error occurred.

See also: ReadComponentRes (303), ReadComponentResEx (303), ReadComponentResFile (303)

4.4.25 IntToIdent

Synopsis: Looks up an identifier for an integer value in a identifier-to-integer map list.

Declaration: `function IntToIdent(Int: LongInt; var Ident: string; const Map: Array of TIdentMapEntry) : Boolean`

Visibility: default

Description: `IdentToInt` searches `Map` for an entry whose `Value` field matches `Int` and returns the corresponding identifier in `Ident`. If a match was found, the function returns `True`, otherwise, `False` is returned.

See also: TIdentToInt (281), TintToIdent (282), IdentToInt (299), TIdentMapEntry (309)

4.4.26 InvalidPoint

Synopsis: Check whether a point is invalid.

Declaration:
```
function InvalidPoint(X: Integer; Y: Integer) : Boolean
function InvalidPoint(const At: TPoint) : Boolean
function InvalidPoint(const At: TSmallPoint) : Boolean
```

Visibility: default

Description: `InvalidPoint` returns `True` if the `X` and `Y` coordinates (of the `TPoint` or `TSmallPoint` records, if one of these versions is used) are -1.

See also: TPoint (284), TSmallPoint (287), PointsEqual (302)

4.4.27 LineStart

Synopsis: Finds the start of a line in `Buffer` before `BufPos`.

Declaration:
```
function LineStart(Buffer: PChar; BufPos: PChar) : PChar
```

Visibility: default

Description: `LineStart` reversely scans `Buffer` starting at `BufPos` for a linefeed character. It returns a pointer at the linefeed character.

4.4.28 NotifyGlobalLoading

Synopsis: Not yet implemented.

Declaration:
```
procedure NotifyGlobalLoading
```

Visibility: default

Description: Not yet implemented.

4.4.29 ObjectBinaryToText

Synopsis: Converts an object stream from a binary to a text format.

Declaration:
```
procedure ObjectBinaryToText(Input: TStream; Output: TStream;
                             Encoding: TObjectTextEncoding)
procedure ObjectBinaryToText(Input: TStream; Output: TStream)
```

Visibility: default

Description: `ObjectBinaryToText` reads an object stream in binary format from `Input` and writes the object stream in text format to `Output`. No components are instantiated during the process, this is a pure conversion routine.

See also: ObjectTextToBinary (302)

4.4.30 ObjectResourceToText

Synopsis: Converts an object stream from a (windows) resource to a text format.

Declaration:
```
procedure ObjectResourceToText(Input: TStream; Output: TStream)
```

Visibility: default

Description: `ObjectResourceToText` reads the resource header from the `Input` stream and then passes the streams to ObjectBinaryToText (301)

See also: ObjectBinaryToText (301), ObjectTextToResource (302)

4.4.31 ObjectTextToBinary

Synopsis: Converts an object stream from a text to a binary format.

Declaration:
```
procedure ObjectTextToBinary(Input: TStream; Output: TStream)
```

Visibility: default

Description: Converts an object stream from a text to a binary format.

4.4.32 ObjectTextToResource

Synopsis: Converts an object stream from a text to a (windows) resource format.

Declaration:
```
procedure ObjectTextToResource(Input: TStream; Output: TStream)
```

Visibility: default

Description: `ObjectTextToResource` reads an object stream in text format from `Input` and writes a resource stream to `Output`.

Note that for the current implementation of this method in Free Pascal, the output stream should support positioning. (e.g. it should not be a pipe)

See also: ObjectBinaryToText (301), ObjectResourceToText (302)

4.4.33 Point

Synopsis: Returns a `TPoint` record with the given coordinates.

Declaration:
```
function Point(AX: Integer; AY: Integer) : TPoint
```

Visibility: default

Description: `Point` returns a TPoint (284) record with the given coordinates `AX` and `AY` filled in.

See also: TPoint (284), SmallPoint (307), Rect (304), Bounds (295)

4.4.34 PointsEqual

Synopsis: Check whether two `TPoint` variables are equal.

Declaration:
```
function PointsEqual(const P1: TPoint; const P2: TPoint) : Boolean
function PointsEqual(const P1: TSmallPoint; const P2: TSmallPoint)
                     : Boolean
```

Visibility: default

Description: `PointsEqual` compares the `P1` and `P2` points (of type TPoint (284) or TSmallPoint (287)) and returns `True` if the `X` and `Y` coordinates of the points are equal, or `False` otherwise.

See also: TPoint (284), TSmallPoint (287), InvalidPoint (301)

4.4.35 ReadComponentRes

Synopsis: Read component properties from a resource in the current module.

Declaration:
```
function ReadComponentRes(const ResName: string; Instance: TComponent)
                                  : TComponent
```

Visibility: default

Description: `ReadComponentRes` will read the component's properties from the resource `ResName` in the current module (always program module). It returns Nil if the resource was not found. It returns `Instance` if the resource was found and successfully applied to the component.

Errors: The function may raise an exception if the stream contains wrong data.

See also: ReadComponentResEx (303)

4.4.36 ReadComponentResEx

Synopsis: Read component properties from a resource in the specified module.

Declaration:
```
function ReadComponentResEx(HInstance: THandle; const ResName: string)
                                  : TComponent
```

Visibility: default

Description: `ReadComponentRes` will locate the resource `ResName` in instance `HInstance` (the current program, normally). It returns `Nil` if the resource was not found. It returns an instantiated component with all properties found in the stream, applied. This requires that the component is registered using registerclass.

Errors: The function may raise an exception if the stream contains wrong data.

See also: ReadComponentRes (303)

4.4.37 ReadComponentResFile

Synopsis: Read component properties from a specified resource file.

Declaration:
```
function ReadComponentResFile(const FileName: string;
                                  Instance: TComponent) : TComponent
```

Visibility: default

Description: `ReadComponentResFile` starts reading properties for `Instance` from the file `FileName`. It creates a filestream from `FileName` and then calls the TStream.ReadComponentRes (459) method to read the state of the component from the stream.

See also: TStream.ReadComponentRes (459), WriteComponentResFile (309)

4.4.38 Rect

Synopsis: Returns a `TRect` record with the given coordinates.

Declaration:
```
function Rect(ALeft: Integer; ATop: Integer; ARight: Integer;
              ABottom: Integer) : TRect
```

Visibility: default

Description: `Rect` returns a TRect (285) record with the given top-left `(ALeft,ATop)` and bottom-right `(ABottom,ARight)` corners filled in.

No checking is done to see whether the coordinates are valid.

See also: TRect (285), Point (302), SmallPoint (307), Bounds (295)

4.4.39 RedirectFixupReferences

Synopsis: Redirects references under the `root` object from `OldRootName` to `NewRootName`.

Declaration:
```
procedure RedirectFixupReferences(Root: TComponent;
                                  const OldRootName: string;
                                  const NewRootName: string)
```

Visibility: default

Description: `RedirectFixupReferences` examines the list of unresolved references and replaces references to a root object named `OldRootName` with references to root object `NewRootName`.

An application programmer should never need to call `RedirectFixupReferences`. This function can be used by an IDE to support redirection of broken component links.

See also: RemoveFixupReferences (307)

4.4.40 RegisterClass

Synopsis: Registers a class with the streaming system.

Declaration:
```
procedure RegisterClass(AClass: TPersistentClass)
```

Visibility: default

Description: `RegisterClass` registers the class `AClass` in the streaming system. After the class has been registered, it can be read from a stream when a reference to this class is encountered.

See also: RegisterClasses (305), RegisterClassAlias (304), RegisterComponents (305), UnregisterClass (308)

4.4.41 RegisterClassAlias

Synopsis: Registers a class alias with the streaming system.

Declaration:
```
procedure RegisterClassAlias(AClass: TPersistentClass;
                             const Alias: string)
```

Visibility: default

Description: `RegisterClassAlias` registers a class alias in the streaming system. If a reference to a class `Alias` is encountered in a stream, then an instance of the class `AClass` will be created instead by the streaming code.

See also: RegisterClass (304), RegisterClasses (305), RegisterComponents (305), UnregisterClass (308)

4.4.42 RegisterClasses

Synopsis: Registers multiple classes with the streaming system.

Declaration:
```
procedure RegisterClasses(AClasses: Array of TPersistentClass)
```

Visibility: default

Description: `RegisterClasses` registers the specified classes `AClass` in the streaming system. After the classes have been registered, they can be read from a stream when a reference to this class is encountered.

See also: RegisterClass (304), RegisterClassAlias (304), RegisterComponents (305), UnregisterClass (308)

4.4.43 RegisterComponents

Synopsis: Registers components for the component palette.

Declaration:
```
procedure RegisterComponents(const Page: string;
                             ComponentClasses: Array of TComponentClass)
```

Visibility: default

Description: `RegisterComponents` registers the component on the appropriate component page. The component pages can be used by an IDE to display the known components so an application programmer may pick and use the components in his programs.

Registercomponents inserts the component class in the correct component page. If the `RegisterComponentsProc` procedure is set, this is called as well. Note that this behaviour is different from Delphi's behaviour where an exception will be raised if the procedural variable is not set.

See also: RegisterClass (304), RegisterNoIcon (306)

4.4.44 RegisterFindGlobalComponentProc

Synopsis: Register a component searching handler.

Declaration:
```
procedure RegisterFindGlobalComponentProc
                                        (AFindGlobalComponent: TFindGlobalComponent
```

Visibility: default

Description: `RegisterFindGlobalComponentProc` registers a global component search callback `AFindGlobalComponent`. When FindGlobalComponent (297) is called, then this callback will be used to search for the component.

Errors: None.

See also: FindGlobalComponent (297), UnRegisterFindGlobalComponentProc (308)

4.4.45 RegisterInitComponentHandler

Synopsis: Register a component initialization handler.

Declaration:
```
procedure RegisterInitComponentHandler(ComponentClass: TComponentClass;
                                       Handler: TInitComponentHandler)
```

Visibility: default

Description: `RegisterInitComponentHandler` registers a component initialization handler `Handler` for the component `ComponentClass`. This handler will be used to initialize descendants of `ComponentClass` in the InitInheritedComponent (300) call.

See also: InitInheritedComponent (300), TInitComponentHandler (282)

4.4.46 RegisterIntegerConsts

Synopsis: Registers some integer-to-identifier mappings.

Declaration:
```
procedure RegisterIntegerConsts(IntegerType: Pointer;
                                IdentToIntFn: TIdentToInt;
                                IntToIdentFn: TIntToIdent)
```

Visibility: default

Description: `RegisterIntegerConsts` registers a pair of callbacks to be used when an integer of type `IntegerType` must be mapped to an identifier (using `IntToIdentFn`) or when an identifier must be mapper to an integer (using `IdentToIntFn`).

Component programmers can use `RegisterIntegerConsts` to associate a series of identifier strings with integer values for a property. A necessary condition is that the property should have a separate type declared using the `type integer` syntax. If a type of integer is defined in this way, an IDE can show symbolic names for the values of these properties.

The `IntegerType` should be a pointer to the type information of the integer type. The `IntToIdentFn` and `IdentToIntFn` are two callbacks that will be used when converting between the identifier and integer value and vice versa. The functions IdentToInt (299) and IntToIdent (300) can be used to implement these callback functions.

See also: TIdentToInt (281), TIntToIdent (282), IdentToInt (299), IntToIdent (300)

4.4.47 RegisterNoIcon

Synopsis: Registers components that have no icon on the component palette.

Declaration:
```
procedure RegisterNoIcon(ComponentClasses: Array of TComponentClass)
```

Visibility: default

Description: `RegisterNoIcon` performs the same function as RegisterComponents (305) except that it calls RegisterNoIconProc (294) instead of RegisterComponentsProc (294)

See also: RegisterNoIconProc (294), RegisterComponents (305)

4.4.48 RegisterNonActiveX

Synopsis: Register non-activex component.

Declaration:
```
procedure RegisterNonActiveX
                              (ComponentClasses: Array of TComponentClass;
                              AxRegType: TActiveXRegType)
```

Visibility: default

Description: Not yet implemented in Free Pascal

4.4.49 RemoveFixupReferences

Synopsis: Removes references to rootname from the fixup list.

Declaration:
```
procedure RemoveFixupReferences(Root: TComponent;
                                const RootName: string)
```

Visibility: default

Description: `RemoveFixupReferences` examines the list of unresolved references and removes references to a root object pointing at `Root` or a root component named `RootName`.

An application programmer should never need to call `RemoveFixupReferences`. This function can be used by an IDE to support removal of broken component links.

See also: RedirectFixupReferences (304)

4.4.50 RemoveFixups

Synopsis: Removes `Instance` from the fixup list.

Declaration:
```
procedure RemoveFixups(Instance: TPersistent)
```

Visibility: default

Description: `RemoveFixups` removes all entries for component `Instance` from the list of unresolved references.a

See also: RedirectFixupReferences (304), RemoveFixupReferences (307)

4.4.51 SmallPoint

Synopsis: Returns a `TSmallPoint` record with the given coordinates.

Declaration:
```
function SmallPoint(AX: SmallInt; AY: SmallInt) : TSmallPoint
```

Visibility: default

Description: `SmallPoint` returns a TSmallPoint (287) record with the given coordinates `AX` and `AY` filled in.

See also: TSmallPoint (287), Point (302), Rect (304), Bounds (295)

4.4.52 StartClassGroup

Synopsis: Start new class group.

Declaration:
```
procedure StartClassGroup(AClass: TPersistentClass)
```

Visibility: default

Description: `StartClassGroup` starts a new class group and adds `AClass` to it.

The class registration and streaming mechanism allows to organize the classes in groups. This allows an IDE to form groups of classes, which can be enabled or disabled. It is not needed at Run-Time.

See also: GroupDescendentsWith (299), ActivateClassGroup (294), ClassGroupOf (296)

4.4.53 UnRegisterClass

Synopsis: Unregisters a class from the streaming system.

Declaration:
```
procedure UnRegisterClass(AClass: TPersistentClass)
```

Visibility: default

Description: `UnregisterClass` removes the class `AClass` from the class definitions in the streaming system.

See also: UnRegisterClasses (308), UnRegisterModuleClasses (308), RegisterClass (304)

4.4.54 UnRegisterClasses

Synopsis: Unregisters multiple classes from the streaming system.

Declaration:
```
procedure UnRegisterClasses(AClasses: Array of TPersistentClass)
```

Visibility: default

Description: `UnregisterClasses` removes the classes in `AClasses` from the class definitions in the streaming system.

4.4.55 UnregisterFindGlobalComponentProc

Synopsis: Remove a previously registered component searching handler.

Declaration:
```
procedure UnregisterFindGlobalComponentProc
                                              (AFindGlobalComponent: TFindGlobalCompone
```

Visibility: default

Description: `UnRegisterFindGlobalComponentProc` unregisters the previously registered global component search callback `AFindGlobalComponent`. After this call, when FindGlobalComponent (297) is called, then this callback will be no longer be used to search for the component.

Errors: None.

See also: FindGlobalComponent (297), RegisterFindGlobalComponentProc (305)

4.4.56 UnRegisterModuleClasses

Synopsis: Unregisters classes registered by module.

Declaration:
```
procedure UnRegisterModuleClasses(Module: HMODULE)
```

Visibility: default

Description: `UnRegisterModuleClasses` unregisters all classes which reside in the module `Module`. For each registered class, the definition pointer is checked to see whether it resides in the module, and if it does, the definition is removed.

See also: UnRegisterClass (308), UnRegisterClasses (308), RegisterClasses (305)

4.4.57 WriteComponentResFile

Synopsis: Write component properties to a specified resource file.

Declaration:
```
procedure WriteComponentResFile(const FileName: string;
                                Instance: TComponent)
```

Visibility: default

Description: `WriteComponentResFile` starts writing properties of `Instance` to the file `FileName`. It creates a filestream from `FileName` and then calls TStream.WriteComponentRes (460) method to write the state of the component to the stream.

See also: TStream.WriteComponentRes (460), ReadComponentResFile (303)

4.5 TIdentMapEntry

```
TIdentMapEntry = record
  Value : Integer;
  Name : string;
end
```

`TIdentMapEntry` is used internally by the IdentToInt (299) and IntToIdent (300) calls to store the mapping between the identifiers and the integers they represent.

4.6 TStringItem

```
TStringItem = record
  FString : string;
  FObject : TObject;
end
```

The `TStringItem` is used to store the string and object items in a TStringList (470) string list instance. It should never be used directly.

4.7 EBitsError

4.7.1 Description

When an index of a bit in a TBits (358) is out of the valid range (`0` to `Count-1`) then a `EBitsError` exception is raised.

4.8 EClassNotFound

4.8.1 Description

When the streaming system needs to create a component, it looks for the class pointer (VMT) in the list of registered classes by its name. If this name is not found, then an `EClassNotFound` is raised.

See also: EFilerError (310)

4.9 EComponentError

4.9.1 Description

When an error occurs during the registration of a component, or when naming a component, then a `EComponentError` is raised. Possible causes are:

1. An name with an illegal character was assigned to a component.
2. A component with the same name and owner already exists.
3. The component registration system isn't set up properly.

See also: TComponent (376), TComponent.Name (385)

4.10 EFCreateError

4.10.1 Description

When the operating system reports an error during creation of a new file in the Filestream Constructor (396), a `EFCreateError` is raised.

See also: EStreamError (312), EFOpenError (310)

4.11 EFilerError

4.11.1 Description

This class serves as an ancestor class for exceptions that are raised when an error occurs during component streaming. A `EFilerError` exception is raised when a class is registered twice.

See also: EStreamError (312), EReadError (311)

4.12 EFOpenError

4.12.1 Description

When the operating system reports an error during the opening of a file in the Filestream Constructor (396), a `EFOpenError` is raised.

See also: EStreamError (312), EFCreateError (310)

4.13 EInvalidImage

4.13.1 Description

This exception is not used by Free Pascal but is provided for Delphi compatibility.

4.14 EInvalidOperation

4.14.1 Description

This exception is not used in Free Pascal, it is defined for Delphi compatibility purposes only.

4.15 EMethodNotFound

4.15.1 Description

This exception is no longer used in the streaming system. This error is replaced by a EReadError (311).

See also: EFilerError (310), EReadError (311)

4.16 EObserver

4.16.1 Description

`EObserver` is an error that is raised when an object is registered as an observer, and it does not implement the IFPObserver (315) interface.

See also: IFPObserver (315), IFPObserved.FPOAttachObserver (314)

4.17 EOutOfResources

4.17.1 Description

This exception is not used in Free Pascal, it is defined for Delphi compatibility purposes only.

4.18 EParserError

4.18.1 Description

When an error occurs during the parsing of a stream, an `EParserError` is raised. Usually this indicates that an invalid token was found on the input stream, or the token read from the stream wasn't the expected token.

See also: TParser (429)

4.19 EReadError

4.19.1 Description

If an error occurs when reading from a stream, a `EReadError` exception is raised. Possible causes for this are:

1. Not enough data is available when reading from a stream

2. The stream containing a component's data contains invalid data. this will occur only when reading a component from a stream.

See also: EFilerError (310), EWriteError (313)

4.20 EResNotFound

4.20.1 Description

This exception is not used by Free Pascal but is provided for Delphi compatibility.

4.21 EStreamError

4.21.1 Description

An `EStreamError` is raised when an error occurs during reading from or writing to a stream: Possible causes are

1. Not enough data is available in the stream.
2. Trying to seek beyond the beginning or end of the stream.
3. Trying to set the capacity of a memory stream and no memory is available.
4. Trying to write to a read-only stream, such as a resource stream.
5. Trying to read from a write-only stream.

See also: EFCreateError (310)

4.22 EStringListError

4.22.1 Description

When an error occurs in one of the methods of TStrings (475) then an `EStringListError` is raised. This can have one of the following causes:

1. There is not enough memory to expand the list.
2. The list tried to grow beyond its maximal capacity.
3. A non-existent element of the list was referenced. (i.e. the list index was out of bounds)
4. An attempt was made to add a duplicate entry to a TStringList (470) when TStringList.Duplicates (473) is `False`.

See also: TStrings (475), TStringList (470)

4.23 EThread

4.23.1 Description

Thread error exception.

4.24 EThreadDestroyCalled

4.24.1 Description

Exception raised when a thread is destroyed illegally.

4.25 EThreadExternalException

4.25.1 Description

`EThreadExternalException` is raised by for example TThread.CheckTerminated (510) and TThread.SetReturnValue (510) when the thread was not created by the Free Pascal program, but by an external code base (for example a DLL, or the calling application in a DLL).

See also: TThread.CheckTerminated (510), TThread.SetReturnValue (510)

4.26 EWriteError

4.26.1 Description

If an error occurs when writing to a stream, a `EWriteError` exception is raised. Possible causes for this are:

1. The stream doesn't allow writing.
2. An error occurred when writing a property to a stream.

See also: EFilerError (310), EReadError (311)

4.27 IDesignerNotify

4.27.1 Description

`IDesignerNotify` is an interface that can be used to communicate changes to a designer mechanism. It offers functionality for detecting changes, and notifications when the component is destroyed.

4.27.2 Method overview

Page	Method	Description
313	Modified	Notify that the component is modified.
314	Notification	Notification of owner changes.

4.27.3 IDesignerNotify.Modified

Synopsis: Notify that the component is modified.

Declaration: `procedure Modified`

Visibility: default

Description: `Modified` can be used to notify a designer of changes, indicating that components should be streamed.

4.27.4 IDesignerNotify.Notification

Synopsis: Notification of owner changes.

Declaration: `procedure Notification(AnObject: TPersistent; Operation: TOperation)`

Visibility: default

Description: `Notification` is the interface counterpart of TComponent.Notification (377) which is used to communicate adds to the components.

See also: TComponent.Notification (377)

4.28 IFPObserved

4.28.1 Description

`IFPObserved` is an interface which can be implemented in objects that must be observable. Objects that wish to observe the object can register themselves with the FPOAttachObserver (272) call, and must be detached using the FPODetachObserver (272) call.

This interface is not reference counted, so care must be taken that the `ooFree` message is sent with FPONotifyObservers (272) when the object is freed.

See also: FPONotifyObservers (272)

4.28.2 Method overview

Page	Method	Description
314	FPOAttachObserver	Attach a new observer to the object.
315	FPODetachObserver	Remove an observer from the list of observers.
315	FPONotifyObservers	Notify all observers.

4.28.3 IFPObserved.FPOAttachObserver

Synopsis: Attach a new observer to the object.

Declaration: `procedure FPOAttachObserver(AObserver: TObject)`

Visibility: default

Description: `FPOAttachObserver` must be called with an object instance `AObserver` that implements the IFPObserver (315) interface. The FPOObservedChanged (316) method of the interface will be called whenever FPONotifyObservers (272) is used to notify observers of a change. Objects implementing this interface should check that `AObserver` actually implements the IFPObserver (315) interface.

Do not make assumptions on how the interface behaves if `FPOAttachObserver` is called more than once with the same interface. It may add the object to the list of observers unconditionally (in which case it will be notified twice) or it may check that it is not yet in the list.

Errors: If `AObserver` does not implement the IFPObserver (315) interface, an EObserver (311) exception must be raised. No other errors should be raised, other than a possible out of memory error.

See also: IFPObserver (315), FPOObservedChanged (316), FPONotifyObservers (272)

4.28.4 IFPObserved.FPODetachObserver

Synopsis: Remove an observer from the list of observers.

Declaration:
```
procedure FPODetachObserver(AObserver: TObject)
```

Visibility: default

Description: `FPODetachObserver` removes the `AObserver` object from the list of observers. If it was not in the list, then this is silently accepted. Once removed, it will no longer receive notifications when FPOObservedChanged (316) is called.

If the object was added more than once using FPOAttachObserver (272), then it depends on the implementor of the interface whether or `FPODetachObserver` must be called an equal number of times.

See also: IFPObserver (315), FPOObservedChanged (316), FPONotifyObservers (272), FPOAttachObserver (272)

4.28.5 IFPObserved.FPONotifyObservers

Synopsis: Notify all observers.

Declaration:
```
procedure FPONotifyObservers(ASender: TObject;
                             AOperation: TFPObservedOperation;
                             Data: Pointer)
```

Visibility: default

Description: `FPONotifyObservers` notifies all observers of the object that a change has occurred. It calls FPOObservedChanged (316) on the IFPObserver (315) interface of all attached objects, and passes on `ASender` (normally this is `Self`), `AOperation` and `Data`. What `Data` is, depends on the implementor of the interface.

There is no guaranteed order in which the change notifications are delivered to the observers. This is an implementation-specific detail, which should not be relied upon in any way.

See also: IFPObserver (315), FPOObservedChanged (316), FPODetachObserver (272), FPOAttachObserver (272)

4.29 IFPObserver

4.29.1 Description

`IFPObserver` is the interface an object must implement if it wishes to receive change notifications from another object. The presence of this interface will be checked when the object registers itself using IFPObserver.FPOAttachObserver (315). The change notifications arrive because the FPOObservedChanged (272) method is called by the observed object.

See also: IFPObserved (314), FPOAttachObserver (314)

4.29.2 Method overview

Page	Method	Description
316	FPOObservedChanged	Entry point for change notifications.

4.29.3 IFPObserver.FPOObservedChanged

Synopsis: Entry point for change notifications.

Declaration:
```
procedure FPOObservedChanged(ASender: TObject;
                             Operation: TFPObservedOperation;
                             Data: Pointer)
```

Visibility: default

Description: `FPOObservedChanged` is the method that is called by an observed object (IFPObserved (314)) when it calls FPONotifyObservers (315) The `Sender` is the object under observation, the `Operation` and `Data` are the parameters used in the call to `FPONotifyObservers`.

See also: IFPObserved (314), FPONotifyObservers (315)

4.30 IInterfaceComponentReference

4.30.1 Description

`IInterfaceComponentReference` is an interface to return the component that implements a given interface. It is implemented by TComponent (376).

See also: TComponent (376)

4.30.2 Method overview

Page	Method	Description
316	GetComponent	Return component instance.

4.30.3 IInterfaceComponentReference.GetComponent

Synopsis: Return component instance.

Declaration:
```
function GetComponent : TComponent
```

Visibility: default

Description: `GetComponent` returns the component instance.

Errors: None.

See also: TComponent (376)

4.31 IInterfaceList

4.31.1 Description

`IInterfaceList` is an interface for maintaining a list of interfaces, strongly resembling the standard TList (415) class. It offers the same list of public methods as `TList`, with the exception that it uses interfaces instead of pointers.

All interfaces in the list should descend from `IUnknown`.

More detailed descriptions of how the various methods behave can be found in the `TList` reference.

See also: TList (415)

4.31.2 Method overview

Page	Method	Description
320	Add	Add an interface to the list.
319	Clear	Clear the list.
319	Delete	Remove an interface from the list.
319	Exchange	Exchange 2 interfaces in the list.
319	First	Return the first non-empty interface in the list.
317	Get	Retrieve an interface pointer from the list.
317	GetCapacity	Return the capacity of the list.
318	GetCount	Return the current number of elements in the list.
320	IndexOf	Return the index of an interface.
320	Insert	Insert an interface in the list.
320	Last	Returns the last non-nil interface in the list.
321	Lock	Lock the list.
318	Put	Write an item to the list.
320	Remove	Remove an interface from the list.
318	SetCapacity	Set the capacity of the list.
318	SetCount	Set the number of items in the list.
321	Unlock	Unlock the list.

4.31.3 Property overview

Page	Properties	Access	Description
321	Capacity	rw	Capacity of the list.
321	Count	rw	Current number of elements in the list.
322	Items	rw	Provides Index-based, sequential, access to the interfaces in the list.

4.31.4 IInterfaceList.Get

Synopsis: Retrieve an interface pointer from the list.

Declaration: `function Get(i: Integer) : IUnknown`

Visibility: default

Description: `Get` returns the interface pointer at position `i` in the list. It serves as the `Read` method for the Items (322) property.

See also: IInterfaceList.Items (322), TList.Items (423)

4.31.5 IInterfaceList.GetCapacity

Synopsis: Return the capacity of the list.

Declaration: `function GetCapacity : Integer`

Visibility: default

Description: `GetCapacity` returns the current capacity of the list. It serves as the `Read` method for the Capacity (321) property.

See also: IInterfaceList.Capacity (321), TList.Capacity (422)

4.31.6 IInterfaceList.GetCount

Synopsis: Return the current number of elements in the list.

Declaration: `function GetCount : Integer`

Visibility: default

Description: It serves as the `Read` method for the Count (321) property.

See also: IInterfaceList.Count (321), TList.Count (423)

4.31.7 IInterfaceList.Put

Synopsis: Write an item to the list.

Declaration: `procedure Put(i: Integer; item: IUnknown)`

Visibility: default

Description: `Put` writes the interface `Item` at position `I` in the list. It servers as the `Write` method for the Items (322) property.

See also: IInterfaceList.Items (322), TList.Items (423)

4.31.8 IInterfaceList.SetCapacity

Synopsis: Set the capacity of the list.

Declaration: `procedure SetCapacity(NewCapacity: Integer)`

Visibility: default

Description: `SetCapacity` sets the capacity of the list to `NewCapacity`. It serves as the `Write` method for the Capacity (321) property.

See also: IInterfaceList.Capacity (321), TList.Capacity (422)

4.31.9 IInterfaceList.SetCount

Synopsis: Set the number of items in the list.

Declaration: `procedure SetCount(NewCount: Integer)`

Visibility: default

Description: `SetCount` sets the count of the list to `NewCount`. It serves as the `Write` method for the Capacity (321)

See also: IInterfaceList.Count (321), TList.Count (423)

4.31.10 IInterfaceList.Clear

Synopsis: Clear the list.

Declaration: `procedure Clear`

Visibility: default

Description: `Clear` removes all interfaces from the list. All interfaces in the list will be cleared (i.e. their reference count will decrease with 1)

See also: TList.Clear (418)

4.31.11 IInterfaceList.Delete

Synopsis: Remove an interface from the list.

Declaration: `procedure Delete(index: Integer)`

Visibility: default

Description: `Delete` removes the interface at position `Index` from the list. It does this by explicitly clearing the interface and then removing the slot.

See also: TList.Clear (418), IInterfaceList.Add (320), IInterfaceList.Delete (319), IInterfaceList.Insert (320)

4.31.12 IInterfaceList.Exchange

Synopsis: Exchange 2 interfaces in the list.

Declaration: `procedure Exchange(index1: Integer; index2: Integer)`

Visibility: default

Description: `Exchange` exchanges 2 interfaces in the list at locations `index1` and `Index2`.

See also: TList.Exchange (419), IInterfaceList.Add (320), IInterfaceList.Delete (319), IInterfaceList.Insert (320)

4.31.13 IInterfaceList.First

Synopsis: Return the first non-empty interface in the list.

Declaration: `function First : IUnknown`

Visibility: default

Description: `First` returns the first non-empty interface in the list.

See also: TList.First (420), IInterfaceList.IndexOf (320), IInterfaceList.Last (320)

4.31.14 IInterfaceList.IndexOf

Synopsis: Return the index of an interface.

Declaration: `function IndexOf(const item: IUnknown) : Integer`

Visibility: default

Description: `IndexOf` returns the location in the list of the interface `Item`. If there is no such interface in the list, then -1 is returned.

See also: TList.IndexOf (420), IInterfaceList.First (319), IInterfaceList.Last (320)

4.31.15 IInterfaceList.Add

Synopsis: Add an interface to the list.

Declaration: `function Add(item: IUnknown) : Integer`

Visibility: default

Description: `Add` adds the interface `Item` to the list, and returns the position at which it has been added.

See also: TList.Add (418), IInterfaceList.Insert (320), IInterfaceList.Delete (319)

4.31.16 IInterfaceList.Insert

Synopsis: Insert an interface in the list.

Declaration: `procedure Insert(i: Integer; item: IUnknown)`

Visibility: default

Description: `Insert` inserts the interface `Item` in the list, at position `I`, shifting all items one position.

See also: TList.Insert (420), IInterfaceList.Add (320), IInterfaceList.Delete (319)

4.31.17 IInterfaceList.Last

Synopsis: Returns the last non-nil interface in the list.

Declaration: `function Last : IUnknown`

Visibility: default

Description: `Last` returns the last non-empty interface in the list.

See also: TList.Last (421), IInterfaceList.First (319), IInterfaceList.IndexOf (320)

4.31.18 IInterfaceList.Remove

Synopsis: Remove an interface from the list.

Declaration: `function Remove(item: IUnknown) : Integer`

Visibility: default

Description: `Remove` searches for the first occurrence of `Item` in the list and deletes it.

See also: TList.Remove (421), IInterfaceList.Delete (319), IInterfaceList.IndexOf (320)

4.31.19 IInterfaceList.Lock

Synopsis: Lock the list.

Declaration: `procedure Lock`

Visibility: default

Description: `Lock` locks the list. After a call to lock, the object list can only be accessed by the current thread, until UnLock (321) is called.

See also: TInterfaceList.Lock (412), IInterfaceList.Unlock (321)

4.31.20 IInterfaceList.Unlock

Synopsis: Unlock the list.

Declaration: `procedure Unlock`

Visibility: default

Description: `Unlock` unlocks a locked list. After a call to `UnLock`, other threads are again able to access the list.

See also: TInterfaceList.UnLock (412), IInterfaceList.Lock (321)

4.31.21 IInterfaceList.Capacity

Synopsis: Capacity of the list.

Declaration: `Property Capacity : Integer`

Visibility: default

Access: Read,Write

Description: `Capacity` is the maximum number of elements the list can hold without needing to reallocate memory for the list. It can be set to improve speed when adding a lot of items to the list.

See also: TList.Capacity (422), IInterfaceList.Count (321)

4.31.22 IInterfaceList.Count

Synopsis: Current number of elements in the list.

Declaration: `Property Count : Integer`

Visibility: default

Access: Read,Write

Description: `Count` is the current number of elements in the list. Setting it to a larger number will allocate empty slots. Setting it to a smaller number will clear any interfaces that fall outside the new border.

See also: IInterfaceList.Capacity (321), TList.Count (423)

4.31.23 IInterfaceList.Items

Synopsis: Provides Index-based, sequential, access to the interfaces in the list.

Declaration: `Property Items[index: Integer]: IUnknown; default`

Visibility: default

Access: Read,Write

Description: `Items` is the default property of the interface list and provides index-based array access to the interfaces in the list. Allowed values for `Index` include 0 to `Count-1`

See also: IInterfaceList.Count (321), TList.Items (423)

4.32 IStreamPersist

4.32.1 Description

`IStreamPersist` defines an interface for object persistence streaming to a stream. Any class implementing this interface is expected to be able to save or load it's state from or to a stream.

See also: TPersistent (435), TComponent (376), TStream (455)

4.32.2 Method overview

Page	Method	Description
322	LoadFromStream	Load persistent data from stream.
322	SaveToStream	Save persistent data to stream.

4.32.3 IStreamPersist.LoadFromStream

Synopsis: Load persistent data from stream.

Declaration: `procedure LoadFromStream(Stream: TStream)`

Visibility: default

Description: `LoadFromStream` is the method called when the object should load it's state from the stream `stream`. It should be able to read the data which was written using the `SavetoStream` method.

See also: TPersistent (435), TComponent (376), TStream (455), IStreamPersist.SaveToStream (322)

4.32.4 IStreamPersist.SaveToStream

Synopsis: Save persistent data to stream.

Declaration: `procedure SaveToStream(Stream: TStream)`

Visibility: default

Description: `SaveFromStream` is the method called when the object should load it's state from the stream `stream`. The data written by this method should be readable by the `LoadFromStream` method.

See also: TPersistent (435), TComponent (376), TStream (455), IStreamPersist.LoadFromStream (322)

4.33 IStringsAdapter

4.33.1 Description

Is not yet supported in Free Pascal.

See also: TStrings (475)

4.33.2 Method overview

Page	Method	Description
323	ReferenceStrings	Add a reference to the indicated strings.
323	ReleaseStrings	Release the reference to the strings.

4.33.3 IStringsAdapter.ReferenceStrings

Synopsis: Add a reference to the indicated strings.

Declaration: `procedure ReferenceStrings(S: TStrings)`

Visibility: default

4.33.4 IStringsAdapter.ReleaseStrings

Synopsis: Release the reference to the strings.

Declaration: `procedure ReleaseStrings`

Visibility: default

4.34 IVCLComObject

4.34.1 Description

`IVCLComObject` is used by TComponent to implement the `IUnknown` interface used by COM automation servers. Partially, it is the translation to pascal of the `IDispatch` interface definition by Microsoft. If `TComponent` needs to return an `IUnknown` interface, it creates a `IVCLComObject` interface instead.

See also: TComponent.VCLComObject (385)

4.34.2 Method overview

Page	Method	Description
325	FreeOnRelease	Is called by TComponent.FreeOnRelease.
324	GetIDsOfNames	The `IDispatch:GetIDsOfNames` call for automation servers.
324	GetTypeInfo	The IDispatch:GetTypeInfo call for automation servers.
324	GetTypeInfoCount	The IDispatch:GetTypeInfoCount call for automation servers.
324	Invoke	The `IDispatch:Invoke` call for automation servers.
325	SafeCallException	This method can be invoked if an exception occurs during `Invoke`.

4.34.3 IVCLComObject.GetTypeInfoCount

Synopsis: The IDispatch:GetTypeInfoCount call for automation servers.

Declaration:
```
function GetTypeInfoCount(out Count: Integer) : HRESULT
```

Visibility: default

Description: `GetTypeInfoCount` must return in `Count` either 0 or 1 to indicate that it provides type information (1) or not (0).

Errors: On error, a nonzero (different from S_OK) return value must be returned.

See also: IVCLComObject.GetTypeInfo (324)

4.34.4 IVCLComObject.GetTypeInfo

Synopsis: The IDispatch:GetTypeInfo call for automation servers.

Declaration:
```
function GetTypeInfo(Index: Integer; LocaleID: Integer; out TypeInfo)
                            : HRESULT
```

Visibility: default

Description: `GetTypeInfo` must return the `Index`-th entry in the type information of the component in `TypeInfo`. The `LocaleID` argument can be used to indicate the locale of the caller, as different type information can be returned depending on the locale.

Errors: On error, a nonzero (different from S_OK) return value must be returned.

See also: IVCLComObject.GetTypeInfoCount (324)

4.34.5 IVCLComObject.GetIDsOfNames

Synopsis: The `IDispatch:GetIDsOfNames` call for automation servers.

Declaration:
```
function GetIDsOfNames(const IID: TGuid; Names: Pointer;
                            NameCount: Integer; LocaleID: Integer;
                            DispIDs: Pointer) : HRESULT
```

Visibility: default

Description: `GetIDsOfNames` must return in `DispIDs` the dispatch Ids for the `NameCount` names of the methods listed in `Names`. The `LocaleID` indicates the locale of the caller.

Errors: On error, a nonzero (different from S_OK) return value must be returned.

See also: IVCLComObject.Invoke (324)

4.34.6 IVCLComObject.Invoke

Synopsis: The `IDispatch:Invoke` call for automation servers.

Declaration:
```
function Invoke(DispID: Integer; const IID: TGuid; LocaleID: Integer;
                       Flags: Word; var Params; VarResult: Pointer;
                       ExcepInfo: Pointer; ArgErr: Pointer) : HRESULT
```

Visibility: default

Description: `Invoke` must invoke the method designated by `DispID`. `IID` can be ignored. `LocaleID` is used by the caller to indicate the locale it is using. The `Flags` argument describes the context in which `Invoke` is called: a method, or property getter/setter. The `Params` argument contains the parameters to the call. The result should be in `VarResult`. On error, `ExcepInfo` and `ArgError` should be filled.

The function should return 0 (`S_OK`) if all went well.

See also: IVCLComObject.GetIDsOfNames (324)

4.34.7 IVCLComObject.SafeCallException

Synopsis: This method can be invoked if an exception occurs during `Invoke`.

Declaration:
```
function SafeCallException(ExceptObject: TObject;
                           ExceptAddr: CodePointer) : HRESULT
```

Visibility: default

Description: `SafeCallException` is called to handle an exception during invocation of the `Invoke` method. The `TObject` implementation of this method returns `E_UNEXPECTED`.

See also: IVCLComObject.Invoke (324)

4.34.8 IVCLComObject.FreeOnRelease

Synopsis: Is called by TComponent.FreeOnRelease.

Declaration:
```
procedure FreeOnRelease
```

Visibility: default

Description: `FreeOnRelease` is called by TComponent.FreeOnRelease (380) for the `IVCLComObject` interface implemented by `TComponent`.

See also: TComponent.FreeOnRelease (380)

4.35 TAbstractObjectReader

4.35.1 Description

The Free Pascal streaming mechanism, while compatible with Delphi's mechanism, differs from it in the sense that the streaming mechanism uses a driver class when streaming components. The `TAbstractObjectReader` class is the base driver class for reading property values from streams. It consists entirely of abstract methods, which must be implemented by descendant classes.

Different streaming mechanisms can be implemented by making a descendant from `TAbstractObjectReader`. The TBinaryObjectReader (345) class is such a descendant class, which streams data in binary (Delphi compatible) format.

All methods described in this class, must be implemented by descendant classes.

See also: TBinaryObjectReader (345)

4.35.2 Method overview

Page	Method	Description
327	BeginComponent	Marks the reading of a new component.
327	BeginProperty	Marks the reading of a property value.
327	BeginRootComponent	Starts the reading of the root component.
326	FlushBuffer	Flush the buffer.
326	NextValue	Returns the type of the next value in the stream.
328	Read	Read raw data from stream.
328	ReadBinary	Read binary data from the stream.
329	ReadCurrency	Read a currency value from the stream.
329	ReadDate	Read a date value from the stream.
328	ReadFloat	Read a float value from the stream.
329	ReadIdent	Read an identifier from the stream.
330	ReadInt16	Read a 16-bit integer from the stream.
331	ReadInt32	Read a 32-bit integer from the stream.
331	ReadInt64	Read a 64-bit integer from the stream.
330	ReadInt8	Read an 8-bit integer from the stream.
331	ReadSet	Reads a set from the stream.
332	ReadSignature	Read resource signature.
328	ReadSingle	Read a single (real-type) value from the stream.
332	ReadStr	Read a shortstring from the stream.
332	ReadString	Read a string of type `StringType` from the stream.
333	ReadUnicodeString	Read a Unicode string value.
327	ReadValue	Reads the type of the next value.
332	ReadWideString	Read a widestring value from the stream.
333	SkipComponent	Skip till the end of the component.
333	SkipValue	Skip the current value.

4.35.3 TAbstractObjectReader.FlushBuffer

Synopsis: Flush the buffer.

Declaration: `procedure FlushBuffer;  Virtual`

Visibility: public

Description: `FlushBuffer` flushes the buffer. It is provided for Delphi compatibility, and is not used in FPC.

See also: TFiler.FlushBuffer (394)

4.35.4 TAbstractObjectReader.NextValue

Synopsis: Returns the type of the next value in the stream.

Declaration: `function NextValue : TValueType;  Virtual;  Abstract`

Visibility: public

Description: This function should return the type of the next value in the stream, but should not read the actual value, i.e. the stream position should not be altered by this method. This is used to 'peek' in the stream what value is next.

See also: TAbstractObjectReader.ReadValue (327)

4.35.5 TAbstractObjectReader.ReadValue

Synopsis: Reads the type of the next value.

Declaration:
```
function ReadValue : TValueType;  Virtual;  Abstract
```

Visibility: public

Description: This function returns the type of the next value in the stream and reads it. i.e. after the call to this method, the stream is positioned to read the value of the type returned by this function.

See also: TAbstractObjectReader.ReadValue (327)

4.35.6 TAbstractObjectReader.BeginRootComponent

Synopsis: Starts the reading of the root component.

Declaration:
```
procedure BeginRootComponent;  Virtual;  Abstract
```

Visibility: public

Description: This function can be used to initialize the driver class for reading a component. It is called once at the beginning of the read process, and is immediately followed by a call to BeginComponent (327).

See also: TAbstractObjectReader.BeginComponent (327)

4.35.7 TAbstractObjectReader.BeginComponent

Synopsis: Marks the reading of a new component.

Declaration:
```
procedure BeginComponent(var Flags: TFilerFlags;
                         var AChildPos: Integer;
                         var CompClassName: string; var CompName: string)
                         ;  Virtual;  Abstract
```

Visibility: public

Description: This method is called when the streaming process wants to start reading a new component.

Descendant classes should override this method to read the start of a component new component definition and return the needed arguments. `Flags` should be filled with any flags that were found at the component definition, as well as `AChildPos`. The `CompClassName` should be filled with the class name of the streamed component, and the `CompName` argument should be filled with the name of the component.

`AChildPos` is used to change the ordering in which components appear below their parent component when streaming descendant forms.

See also: TAbstractObjectReader.BeginRootComponent (327), TAbstractObjectReader.BeginProperty (327)

4.35.8 TAbstractObjectReader.BeginProperty

Synopsis: Marks the reading of a property value.

Declaration:
```
function BeginProperty : string;  Virtual;  Abstract
```

Visibility: public

Description: `BeginProperty` is called by the streaming system when it wants to read a new property. The return value of the function is the name of the property which can be read from the stream.

See also: TAbstractObjectReader.BeginComponent (327)

4.35.9 TAbstractObjectReader.Read

Synopsis: Read raw data from stream.

Declaration: `procedure Read(var Buf; Count: LongInt); Virtual; Abstract`

Visibility: public

Description: `Read` is introduced for Delphi compatibility to read raw data from the component stream. This should not be used in production code as it will totally mess up the streaming.

See also: TBinaryObjectReader.Read (348), TReader.Read (444)

4.35.10 TAbstractObjectReader.ReadBinary

Synopsis: Read binary data from the stream.

Declaration: `procedure ReadBinary(const DestData: TMemoryStream); Virtual; Abstract`

Visibility: public

Description: `ReadBinary` is called when binary data should be read from the stream (i.e. after ReadValue (327) returned a value type of `vaBinary`). The data should be stored in the `DestData` memory stream by descendant classes.

See also: TAbstractObjectReader.ReadFloat (328), TAbstractObjectReader.ReadDate (329), TAbstractObjectReader.ReadSingle (328), TAbstractObjectReader.ReadIdent (329), TAbstractObjectReader.ReadInt8 (330), TAbstractObjectReader.ReadInt16 (330), TAbstractObjectReader.ReadInt32 (331), TAbstractObjectReader.ReadInt64 (331), TabstractObjectReader.ReadSet (331), TabstractObjectReader.ReadStr (332), TabstractObjectReader.ReadString (332)

4.35.11 TAbstractObjectReader.ReadFloat

Synopsis: Read a float value from the stream.

Declaration: `function ReadFloat : Extended; Virtual; Abstract`

Visibility: public

Description: `ReadFloat` is called by the streaming system when it wants to read a float from the stream (i.e. after ReadValue (327) returned a value type of `vaExtended`). The return value should be the value of the float.

See also: TAbstractObjectReader.ReadFloat (328), TAbstractObjectReader.ReadDate (329), TAbstractObjectReader.ReadSingle (328), TAbstractObjectReader.ReadIdent (329), TAbstractObjectReader.ReadInt8 (330), TAbstractObjectReader.ReadInt16 (330), TAbstractObjectReader.ReadInt32 (331), TAbstractObjectReader.ReadInt64 (331), TabstractObjectReader.ReadSet (331), TabstractObjectReader.ReadStr (332), TabstractObjectReader.ReadString (332)

4.35.12 TAbstractObjectReader.ReadSingle

Synopsis: Read a single (real-type) value from the stream.

Declaration: `function ReadSingle : Single; Virtual; Abstract`

Visibility: public

Description: `ReadSingle` is called by the streaming system when it wants to read a single-type float from the stream (i.e. after ReadValue (327) returned a value type of `vaSingle`). The return value should be the value of the float.

See also: TAbstractObjectReader.ReadFloat (328), TAbstractObjectReader.ReadDate (329), TAbstractObjectReader.ReadSingle (328), TAbstractObjectReader.ReadIdent (329), TAbstractObjectReader.ReadInt8 (330), TAbstractObjectReader.ReadInt16 (330), TAbstractObjectReader.ReadInt32 (331), TAbstractObjectReader.ReadInt64 (331), TabstractObjectReader.ReadSet (331), TabstractObjectReader.ReadStr (332), TabstractObjectReader.ReadString (332)

4.35.13 TAbstractObjectReader.ReadDate

Synopsis: Read a date value from the stream.

Declaration: `function ReadDate : TDateTime; Virtual; Abstract`

Visibility: public

Description: `ReadDate` is called by the streaming system when it wants to read a date/time value from the stream (i.e. after ReadValue (327) returned a value type of `vaDate`). The return value should be the date/time value. (This value can be stored as a float, since `TDateTime` is nothing but a float.)

See also: TAbstractObjectReader.ReadFloat (328), TAbstractObjectReader.ReadSingle (328), TAbstractObjectReader.ReadIdent (329), TAbstractObjectReader.ReadInt8 (330), TAbstractObjectReader.ReadInt16 (330), TAbstractObjectReader.ReadInt32 (331), TAbstractObjectReader.ReadInt64 (331), TabstractObjectReader.ReadSet (331), TabstractObjectReader.ReadStr (332), TabstractObjectReader.ReadString (332)

4.35.14 TAbstractObjectReader.ReadCurrency

Synopsis: Read a currency value from the stream.

Declaration: `function ReadCurrency : Currency; Virtual; Abstract`

Visibility: public

Description: `ReadCurrency` is called when a currency-typed value should be read from the stream. This abstract method should be overridden by descendant classes, and should return the currency value read from the stream.

See also: TAbstractObjectWriter.WriteCurrency (337)

4.35.15 TAbstractObjectReader.ReadIdent

Synopsis: Read an identifier from the stream.

Declaration: `function ReadIdent(ValueType: TValueType) : string; Virtual; Abstract`

Visibility: public

Description: `ReadIdent` is called by the streaming system if it expects to read an identifier of type `ValueType` from the stream after a call to ReadValue (327) returned `vaIdent`. The identifier should be returned as a string. Note that in some cases the identifier does not actually have to be in the stream. The following table indicates which identifiers must actually be read:

Table 4.24:

ValueType	Expected value
vaIdent	Read from stream.
vaNil	'Nil'. This does not have to be read from the stream.
vaFalse	'False'. This does not have to be read from the stream.
vaTrue	'True'. This does not have to be read from the stream.
vaNull	'Null'. This does not have to be read from the stream.

See also: TAbstractObjectReader.ReadFloat (328), TAbstractObjectReader.ReadDate (329), TAbstractObjectReader.ReadSingle (328), TAbstractObjectReader.ReadInt8 (330), TAbstractObjectReader.ReadInt16 (330), TAbstractObjectReader.ReadInt32 (331), TAbstractObjectReader.ReadInt64 (331), TabstractObjectReader.ReadSet (331), TabstractObjectReader.ReadStr (332), TabstractObjectReader.ReadString (332)

4.35.16 TAbstractObjectReader.ReadInt8

Synopsis: Read an 8-bit integer from the stream.

Declaration: `function ReadInt8 : ShortInt; Virtual; Abstract`

Visibility: public

Description: `ReadInt8` is called by the streaming process if it expects to read an integer value with a size of 8 bits (1 byte) from the stream (i.e. after ReadValue (327) returned a valuetype of `vaInt8`). The return value is the value if the integer. Note that the size of the value in the stream does not actually have to be 1 byte.

See also: TAbstractObjectReader.ReadFloat (328), TAbstractObjectReader.ReadDate (329), TAbstractObjectReader.ReadSingle (328), TAbstractObjectReader.ReadIdent (329), TAbstractObjectReader.ReadInt16 (330), TAbstractObjectReader.ReadInt32 (331), TAbstractObjectReader.ReadInt64 (331), TabstractObjectReader.ReadSet (331), TabstractObjectReader.ReadStr (332), TabstractObjectReader.ReadString (332)

4.35.17 TAbstractObjectReader.ReadInt16

Synopsis: Read a 16-bit integer from the stream.

Declaration: `function ReadInt16 : SmallInt; Virtual; Abstract`

Visibility: public

Description: `ReadInt16` is called by the streaming process if it expects to read an integer value with a size of 16 bits (2 bytes) from the stream (i.e. after ReadValue (327) returned a valuetype of `vaInt16`). The return value is the value if the integer. Note that the size of the value in the stream does not actually have to be 2 bytes.

See also: TAbstractObjectReader.ReadFloat (328), TAbstractObjectReader.ReadDate (329), TAbstractObjectReader.ReadSingle (328), TAbstractObjectReader.ReadIdent (329), TAbstractObjectReader.ReadInt8 (330), TAbstractObjectReader.ReadInt32 (331), TAbstractObjectReader.ReadInt64 (331), TabstractObjectReader.ReadSet (331), TabstractObjectReader.ReadStr (332), TabstractObjectReader.ReadString (332)

4.35.18 TAbstractObjectReader.ReadInt32

Synopsis: Read a 32-bit integer from the stream.

Declaration: `function ReadInt32 : LongInt; Virtual; Abstract`

Visibility: public

Description: `ReadInt32` is called by the streaming process if it expects to read an integer value with a size of 32 bits (4 bytes) from the stream (i.e. after ReadValue (327) returned a valuetype of `vaInt32`). The return value is the value of the integer. Note that the size of the value in the stream does not actually have to be 4 bytes.

See also: TAbstractObjectReader.ReadFloat (328), TAbstractObjectReader.ReadDate (329), TAbstractObjectReader.ReadSingle (328), TAbstractObjectReader.ReadIdent (329), TAbstractObjectReader.ReadInt8 (330), TAbstractObjectReader.ReadInt16 (330), TAbstractObjectReader.ReadInt64 (331), TAbstractObjectReader.ReadSet (331), TabstractObjectReader.ReadStr (332), TabstractObjectReader.ReadString (332)

4.35.19 TAbstractObjectReader.ReadInt64

Synopsis: Read a 64-bit integer from the stream.

Declaration: `function ReadInt64 : Int64; Virtual; Abstract`

Visibility: public

Description: `ReadInt64` is called by the streaming process if it expects to read an int64 value with a size of 64 bits (8 bytes) from the stream (i.e. after ReadValue (327) returned a valuetype of `vaInt64`). The return value is the value if the integer. Note that the size of the value in the stream does not actually have to be 8 bytes.

See also: TAbstractObjectReader.ReadFloat (328), TAbstractObjectReader.ReadDate (329), TAbstractObjectReader.ReadSingle (328), TAbstractObjectReader.ReadIdent (329), TAbstractObjectReader.ReadInt8 (330), TAbstractObjectReader.ReadInt16 (330), TAbstractObjectReader.ReadInt32 (331), TAbstractObjectReader.ReadSet (331), TabstractObjectReader.ReadStr (332), TabstractObjectReader.ReadString (332)

4.35.20 TAbstractObjectReader.ReadSet

Synopsis: Reads a set from the stream.

Declaration: `function ReadSet(EnumType: Pointer) : Integer; Virtual; Abstract`

Visibility: public

Description: This method is called by the streaming system if it expects to read a set from the stream (i.e. after ReadValue (327) returned a valuetype of `vaSet`). The return value is the contents of the set, encoded in a bitmask the following way:

For each (enumerated) value in the set, the bit corresponding to the ordinal value of the enumerated value should be set. i.e. as `1 shl ord(value)`.

See also: TAbstractObjectReader.ReadFloat (328), TAbstractObjectReader.ReadDate (329), TAbstractObjectReader.ReadSingle (328), TAbstractObjectReader.ReadIdent (329), TAbstractObjectReader.ReadInt8 (330), TAbstractObjectReader.ReadInt16 (330), TAbstractObjectReader.ReadInt32 (331), TAbstractObjectReader.ReadInt64 (331), TabstractObjectReader.ReadStr (332), TabstractObjectReader.ReadString (332)

4.35.21 TAbstractObjectReader.ReadSignature

Synopsis: Read resource signature.

Declaration: `procedure ReadSignature; Virtual; Abstract`

Visibility: public

Description: `ReadSignature` reads the streaming signature from a stream. This method does nothing. It must be implemented by descendents that have a signature header in the stream. (such as binary streams)

See also: TAbstractObjectWriter.WriteSignature (334), TBinaryObjectReader.ReadSignature (350)

4.35.22 TAbstractObjectReader.ReadStr

Synopsis: Read a shortstring from the stream.

Declaration: `function ReadStr : string; Virtual; Abstract`

Visibility: public

Description: `ReadStr` is called by the streaming system if it expects to read a string from the stream (i.e. after ReadValue (327) returned a valuetype of `vaLString`,`vaWstring` or `vaString`). The return value is the string.

See also: TAbstractObjectReader.ReadFloat (328), TAbstractObjectReader.ReadDate (329), TAbstractObjectReader.ReadSingle (328), TAbstractObjectReader.ReadIdent (329), TAbstractObjectReader.ReadInt8 (330), TAbstractObjectReader.ReadInt16 (330), TAbstractObjectReader.ReadInt32 (331), TAbstractObjectReader.ReadInt64 (331), TabstractObjectReader.ReadSet (331), TabstractObjectReader.ReadString (332)

4.35.23 TAbstractObjectReader.ReadString

Synopsis: Read a string of type `StringType` from the stream.

Declaration:
```
function ReadString(StringType: TValueType) : string; Virtual
                        ; Abstract
```

Visibility: public

Description: `ReadStr` is called by the streaming system if it expects to read a string from the stream (i.e. after ReadValue (327) returned a valuetype of `vaLString`,`vaWstring` or `vaString`). The return value is the string.

See also: TAbstractObjectReader.ReadFloat (328), TAbstractObjectReader.ReadDate (329), TAbstractObjectReader.ReadSingle (328), TAbstractObjectReader.ReadIdent (329), TAbstractObjectReader.ReadInt8 (330), TAbstractObjectReader.ReadInt16 (330), TAbstractObjectReader.ReadInt32 (331), TAbstractObjectReader.ReadInt64 (331), TabstractObjectReader.ReadSet (331), TabstractObjectReader.ReadStr (332)

4.35.24 TAbstractObjectReader.ReadWideString

Synopsis: Read a widestring value from the stream.

Declaration: `function ReadWideString : WideString; Virtual; Abstract`

Visibility: public

Description: `ReadWideString` is called when a widestring-typed value should be read from the stream. This abstract method should be overridden by descendant classes.

See also: TAbstractObjectWriter.WriteWideString (338)

4.35.25 TAbstractObjectReader.ReadUnicodeString

Synopsis: Read a Unicode string value.

Declaration: `function ReadUnicodeString : UnicodeString; Virtual; Abstract`

Visibility: public

Description: `ReadUnicodeString` should read a UnicodeString value from the stream. (indicated by the `vaUString` value type).

Descendant classes should override this method to actually read a UnicodeString value.

See also: TBinaryObjectWriter.WriteUnicodeString (357), TAbstractObjectReader.ReadWideString (332)

4.35.26 TAbstractObjectReader.SkipComponent

Synopsis: Skip till the end of the component.

Declaration: `procedure SkipComponent(SkipComponentInfos: Boolean); Virtual ; Abstract`

Visibility: public

Description: This method is used to skip the entire declaration of a component in the stream. Each descendant of `TAbstractObjectReader` should implement this in a way which is optimal for the implemented stream format.

See also: TAbstractObjectReader.BeginComponent (327), TAbstractObjectReader.SkipValue (333)

4.35.27 TAbstractObjectReader.SkipValue

Synopsis: Skip the current value.

Declaration: `procedure SkipValue; Virtual; Abstract`

Visibility: public

Description: `SkipValue` should be used when skipping a value in the stream; The method should determine the type of the value which should be skipped by itself, if this is necessary.

See also: TAbstractObjectReader.SkipComponent (333)

4.36 TAbstractObjectWriter

4.36.1 Description

Abstract driver class for writing component data.

4.36.2 Method overview

Page	Method	Description
334	BeginCollection	Start writing a collection.
334	BeginComponent	Start writing a component.
335	BeginList	Start writing a list.
335	BeginProperty	Start writing a property.
335	EndList	Mark the end of a list.
335	EndProperty	Marks the end of writing of a property.
335	FlushBuffer	Flush the buffer
336	Write	Write raw data to stream.
336	WriteBinary	Writes binary data to the stream.
336	WriteBoolean	Writes a boolean value to the stream.
337	WriteCurrency	Write a currency value to the stream.
337	WriteDate	Writes a date type to the stream.
336	WriteFloat	Writes a float value to the stream.
337	WriteIdent	Writes an identifier to the stream.
337	WriteInteger	Writes an integer value to the stream.
338	WriteMethodName	Writes a methodname to the stream.
338	WriteSet	Writes a set value to the stream.
334	WriteSignature	Write stream signature to the stream.
336	WriteSingle	Writes a single-type real value to the stream.
338	WriteString	Writes a string value to the stream.
337	WriteUInt64	Write an unsigned 64-bit integer.
339	WriteUnicodeString	Write a Unicode string to the stream.
338	WriteVariant	Write a variant to the stream.
338	WriteWideString	Write a widestring value to the stream.

4.36.3 TAbstractObjectWriter.BeginCollection

Synopsis: Start writing a collection.

Declaration:
```
procedure BeginCollection;  Virtual;  Abstract
```

Visibility: public

Description: Start writing a collection.

4.36.4 TAbstractObjectWriter.BeginComponent

Synopsis: Start writing a component.

Declaration:
```
procedure BeginComponent(Component: TComponent; Flags: TFilerFlags;
                         ChildPos: Integer);  Virtual;  Abstract
```

Visibility: public

Description: Start writing a component.

4.36.5 TAbstractObjectWriter.WriteSignature

Synopsis: Write stream signature to the stream.

Declaration:
```
procedure WriteSignature;  Virtual;  Abstract
```

Visibility: public

Description: `WriteSignature` writes the streaming signature to a stream. This method does nothing, it must be implemented by descendents that have a signature header in the stream. (such as binary streams)

See also: TAbstractObjectReader.ReadSignature (332), TBinaryObjectWriter.WriteSignature (353)

4.36.6 TAbstractObjectWriter.BeginList

Synopsis: Start writing a list.

Declaration: `procedure BeginList; Virtual; Abstract`

Visibility: public

Description: Start writing a list.

4.36.7 TAbstractObjectWriter.EndList

Synopsis: Mark the end of a list.

Declaration: `procedure EndList; Virtual; Abstract`

Visibility: public

Description: Mark the end of a list.

4.36.8 TAbstractObjectWriter.BeginProperty

Synopsis: Start writing a property.

Declaration: `procedure BeginProperty(const PropName: string); Virtual; Abstract`

Visibility: public

Description: Start writing a property.

4.36.9 TAbstractObjectWriter.EndProperty

Synopsis: Marks the end of writing of a property.

Declaration: `procedure EndProperty; Virtual; Abstract`

Visibility: public

Description: Marks the end of writing of a property.

4.36.10 TAbstractObjectWriter.FlushBuffer

Synopsis: Flush the buffer

Declaration: `procedure FlushBuffer; Virtual`

Visibility: public

Description: `FlushBuffer` flushes the buffer. It is provided for Delphi compatibility, and is not used in FPC.

See also: TFiler.FlushBuffer (394), TWriter.FlushBuffer (526)

4.36.11 TAbstractObjectWriter.Write

Synopsis: Write raw data to stream.

Declaration: `procedure Write(const Buffer; Count: LongInt); Virtual; Abstract`

Visibility: public

Description: `Write` is introduced for Delphi compatibility to write raw data to the component stream. This should not be used in new production code as it will totally mess up the streaming.

See also: TBinaryObjectWriter.Write (355), TWriter.Write (526)

4.36.12 TAbstractObjectWriter.WriteBinary

Synopsis: Writes binary data to the stream.

Declaration: `procedure WriteBinary(const Buffer; Count: LongInt); Virtual; Abstract`

Visibility: public

Description: Writes binary data to the stream.

4.36.13 TAbstractObjectWriter.WriteBoolean

Synopsis: Writes a boolean value to the stream.

Declaration: `procedure WriteBoolean(Value: Boolean); Virtual; Abstract`

Visibility: public

Description: Writes a boolean value to the stream.

4.36.14 TAbstractObjectWriter.WriteFloat

Synopsis: Writes a float value to the stream.

Declaration: `procedure WriteFloat(const Value: Extended); Virtual; Abstract`

Visibility: public

Description: Writes a float value to the stream.

4.36.15 TAbstractObjectWriter.WriteSingle

Synopsis: Writes a single-type real value to the stream.

Declaration: `procedure WriteSingle(const Value: Single); Virtual; Abstract`

Visibility: public

Description: Writes a single-type real value to the stream.

4.36.16 TAbstractObjectWriter.WriteDate

Synopsis: Writes a date type to the stream.

Declaration: `procedure WriteDate(const Value: TDateTime); Virtual; Abstract`

Visibility: public

Description: Writes a date type to the stream.

4.36.17 TAbstractObjectWriter.WriteCurrency

Synopsis: Write a currency value to the stream.

Declaration: `procedure WriteCurrency(const Value: Currency); Virtual; Abstract`

Visibility: public

Description: `WriteCurrency` is called when a currency-typed value should be written to the stream. This abstract method should be overridden by descendant classes.

See also: TAbstractObjectReader.ReadCurrency (329)

4.36.18 TAbstractObjectWriter.WriteIdent

Synopsis: Writes an identifier to the stream.

Declaration: `procedure WriteIdent(const Ident: string); Virtual; Abstract`

Visibility: public

Description: Writes an identifier to the stream.

4.36.19 TAbstractObjectWriter.WriteInteger

Synopsis: Writes an integer value to the stream.

Declaration: `procedure WriteInteger(Value: Int64); Virtual; Abstract`

Visibility: public

Description: Writes an integer value to the stream.

4.36.20 TAbstractObjectWriter.WriteUInt64

Synopsis: Write an unsigned 64-bit integer.

Declaration: `procedure WriteUInt64(Value: QWord); Virtual; Abstract`

Visibility: public

Description: `WriteUInt64` must be overridden by descendant classes to write a 64-bit unsigned `Value` (valuetype `vaQWord`) to the stream.

Errors: None.

See also: TBinaryObjectWriter.WriteUInt64 (356)

4.36.21 TAbstractObjectWriter.WriteVariant

Synopsis: Write a variant to the stream.

Declaration: `procedure WriteVariant(const Value: Variant); Virtual; Abstract`

Visibility: public

Description: `WriteVariant` must be overridden by descendant classes to write a simple variant type to the stream. `WriteVariant` does not write arrays types or complex types.

See also: TBinaryObjectWriter.WriteVariant (358)

4.36.22 TAbstractObjectWriter.WriteMethodName

Synopsis: Writes a methodname to the stream.

Declaration: `procedure WriteMethodName(const Name: string); Virtual; Abstract`

Visibility: public

Description: Writes a methodname to the stream.

4.36.23 TAbstractObjectWriter.WriteSet

Synopsis: Writes a set value to the stream.

Declaration:
```
procedure WriteSet(Value: LongInt; SetType: Pointer); Virtual
                        ; Abstract
```

Visibility: public

Description: Writes a set value to the stream.

4.36.24 TAbstractObjectWriter.WriteString

Synopsis: Writes a string value to the stream.

Declaration: `procedure WriteString(const Value: string); Virtual; Abstract`

Visibility: public

Description: Writes a string value to the stream.

4.36.25 TAbstractObjectWriter.WriteWideString

Synopsis: Write a widestring value to the stream.

Declaration: `procedure WriteWideString(const Value: WideString); Virtual; Abstract`

Visibility: public

Description: `WriteCurrency` is called when a currency-typed value should be written to the stream. This abstract method should be overridden by descendant classes.

See also: TAbstractObjectReader.ReadWideString (332)

4.36.26 TAbstractObjectWriter.WriteUnicodeString

Synopsis: Write a Unicode string to the stream.

Declaration:
```
procedure WriteUnicodeString(const Value: UnicodeString);  Virtual
                                                          ;  Abstract
```

Visibility: public

Description: `WriteUnicodeString` must be overridden by descendant classes to write a unicodestring (valuetype `vaUString`) value to the stream.

See also: TBinaryObjectWriter.WriteUnicodeString (357)

4.37 TBasicAction

4.37.1 Description

`TBasicAction` implements a basic action class from which all actions are derived. It introduces all basic methods of an action, and implements functionality to maintain a list of clients, i.e. components that are connected with this action.

Do not create instances of `TBasicAction`. Instead, create a descendant class and create an instance of this class instead.

See also: TBasicActionLink (343), TComponent (376)

4.37.2 Method overview

Page	Method	Description
339	Create	Creates a new instance of a TBasicAction (339) class.
340	Destroy	Destroys the action.
341	Execute	Triggers the OnExecute (342) event.
341	ExecuteTarget	Executes the action on the `Target` object.
340	HandlesTarget	Determines whether `Target` can be handled by this action.
341	RegisterChanges	Registers a new client with the action.
342	Suspended	
341	UnRegisterChanges	Unregisters a client from the list of clients.
342	Update	Triggers the OnUpdate (343) event.
340	UpdateTarget	Notify client controls when the action updates itself.

4.37.3 Property overview

Page	Properties	Access	Description
342	ActionComponent	rw	Returns the component that initiated the action.
342	OnExecute	rw	Event triggered when the action executes.
343	OnUpdate	rw	Event triggered when the application is idle.

4.37.4 TBasicAction.Create

Synopsis: Creates a new instance of a TBasicAction (339) class.

Declaration:
```
constructor Create(AOwner: TComponent);  Override
```

Visibility: public

Description: `Create` calls the inherited constructor, and then initializes the list of clients controls (or action lists).

Under normal circumstances it should not be necessary to create a `TBasicAction` descendant manually, actions are created in an IDE.

See also: Destroy (340), AssignClient (339)

4.37.5 TBasicAction.Destroy

Synopsis: Destroys the action.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` cleans up the list of client controls and then calls the inherited destructor.

An application programmer should not call `Destroy` directly; Instead `Free` should be called, if it needs to be called at all. Normally the controlling class (e.g. a TActionList) will destroy the action.

4.37.6 TBasicAction.HandlesTarget

Synopsis: Determines whether `Target` can be handled by this action.

Declaration: `function HandlesTarget(Target: TObject) : Boolean;  Virtual`

Visibility: public

Description: `HandlesTarget` returns `True` if `Target` is a valid client for this action and if so, if it is in a suitable state to execute the action. An application programmer should never need to call `HandlesTarget` directly, it will be called by the action itself when needed.

In `TBasicAction` this method is empty; descendant classes should override this method to implement appropriate checks.

See also: UpdateTarget (340), ExecuteTarget (341)

4.37.7 TBasicAction.UpdateTarget

Synopsis: Notify client controls when the action updates itself.

Declaration: `procedure UpdateTarget(Target: TObject);  Virtual`

Visibility: public

Description: `UpdateTarget` should update the client control specified by `Target` when the action updates itself. In `TBasicAction`, the implementation of `UpdateTarget` is empty. Descendant classes should override and implement `UpdateTarget` to actually update the `Target` object.

An application programmer should never need to call `HandlesTarget` directly, it will be called by the action itself when needed.

See also: HandlesTarget (340), ExecuteTarget (341)

4.37.8 TBasicAction.ExecuteTarget

Synopsis: Executes the action on the `Target` object.

Declaration: `procedure ExecuteTarget(Target: TObject); Virtual`

Visibility: public

Description: `ExecuteTarget` performs the action on the `Target` object. In `TBasicAction` this method does nothing. Descendant classes should implement the action to be performed. For instance an action to post data in a dataset could call the `Post` method of the dataset.

An application programmer should never call `ExecuteTarget` directly.

See also: HandlesTarget (340), UpdateTarget (341), Execute (341)

4.37.9 TBasicAction.Execute

Synopsis: Triggers the OnExecute (342) event.

Declaration: `function Execute : Boolean; Dynamic`

Visibility: public

Description: `Execute` triggers the `OnExecute` event, if one is assigned. It returns `True` if the event handler was called, `False` otherwise.

4.37.10 TBasicAction.RegisterChanges

Synopsis: Registers a new client with the action.

Declaration: `procedure RegisterChanges(Value: TBasicActionLink)`

Visibility: public

Description: `RegisterChanges` adds `Value` to the list of clients.

See also: UnregisterChanges (341)

4.37.11 TBasicAction.UnRegisterChanges

Synopsis: Unregisters a client from the list of clients.

Declaration: `procedure UnRegisterChanges(Value: TBasicActionLink)`

Visibility: public

Description: `UnregisterChanges` removes `Value` from the list of clients. This is called for instance when the action is destroyed, or when the client is assigned a new action.

See also: UnregisterChanges (341), Destroy (340)

4.37.12 TBasicAction.Update

Synopsis: Triggers the OnUpdate (343) event.

Declaration: `function Update : Boolean;  Virtual`

Visibility: public

Description: `Update` triggers the `OnUpdate` event, if one is assigned. It returns `True` if the event was triggered, or `False` if no event was assigned.

Application programmers should never run `Update` directly. The `Update` method is called automatically by the action mechanism; Normally this is in the Idle time of an application. An application programmer should assign the OnUpdate (343) event, and perform any checks in that handler.

See also: OnUpdate (343), Execute (341), UpdateTarget (340)

4.37.13 TBasicAction.Suspended

Declaration: `function Suspended : Boolean;  Virtual`

Visibility: public

4.37.14 TBasicAction.ActionComponent

Synopsis: Returns the component that initiated the action.

Declaration: `Property ActionComponent : TComponent`

Visibility: public

Access: Read,Write

Description: `ActionComponent` is set to the component that caused the action to execute, e.g. a toolbutton or a menu item. The property is set just before the action executes, and is reset to nil after the action was executed.

See also: Execute (341), OnExecute (342)

4.37.15 TBasicAction.OnExecute

Synopsis: Event triggered when the action executes.

Declaration: `Property OnExecute : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnExecute` is the event triggered when the action is activated (executed). The event is triggered e.g. when the user clicks e.g. on a menu item or a button associated to the action. The application programmer should provide a `OnExecute` event handler to execute whatever code is necessary when the button is pressed or the menu item is chosen.

Note that assigning an `OnExecute` handler will result in the Execute (341) method returning a `True` value. Predefined actions (such as dataset actions) will check the result of `Execute` and will not perform their normal task if the `OnExecute` handler was called.

See also: Execute (341), OnUpdate (343)

4.37.16 TBasicAction.OnUpdate

Synopsis: Event triggered when the application is idle.

Declaration: `Property OnUpdate : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnUpdate` is the event triggered when the application is idle, and the action is being updated. The `OnUpdate` event can be used to set the state of the action, for instance disable it if the action cannot be executed at this point in time.

See also: Update (342), OnExecute (342)

4.38 TBasicActionLink

4.38.1 Description

`TBasicActionLink` links an Action to its clients. With each client for an action, a `TBasicActionLink` class is instantiated to handle the communication between the action and the client. It passes events between the action and its clients, and thus presents the action with a uniform interface to the clients.

An application programmer should never use a `TBasicActionLink` instance directly; They are created automatically when an action is associated with a component. Component programmers should create specialized descendants of `TBasicActionLink` which communicate changes in the action to the component.

See also: TBasicAction (339)

4.38.2 Method overview

Page	Method	Description
343	Create	Creates a new instance of the TBasicActionLink class.
344	Destroy	Destroys the TBasicActionLink instance.
344	Execute	Calls the action's Execute method.
344	Update	Calls the action's Update method.

4.38.3 Property overview

Page	Properties	Access	Description
345	Action	rw	The action to which the link was assigned.
345	OnChange	rw	Event handler triggered when the action's properties change.

4.38.4 TBasicActionLink.Create

Synopsis: Creates a new instance of the TBasicActionLink class.

Declaration: `constructor Create(AClient: TObject);  Virtual`

Visibility: public

Description: `Create` creates a new instance of a `TBasicActionLink` and assigns `AClient` as the client of the link.

Application programmers should never instantiate `TBasicActionLink` classes directly. An instance is created automatically when an action is assigned to a control (client).

Component programmers can override the create constructor to initialize further properties.

See also: Destroy (344)

4.38.5 TBasicActionLink.Destroy

Synopsis: Destroys the TBasicActionLink instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` unregisters the `TBasicActionLink` with the action, and then calls the inherited destructor.

Application programmers should never call `Destroy` directly. If a link should be destroyed at all, the `Free` method should be called instead.

See also: Create (343)

4.38.6 TBasicActionLink.Execute

Synopsis: Calls the action's Execute method.

Declaration: `function Execute(AComponent: TComponent) : Boolean;  Virtual`

Visibility: public

Description: `Execute` sets the ActionComponent (342) property of the associated Action (345) to `AComponent` and then calls the Action's execute (341) method. After the action has executed, the `ActionComponent` property is cleared again.

The return value of the function is the return value of the Action's execute method.

Application programmers should never call `Execute` directly. This method will be called automatically when the associated control is activated. (e.g. a button is clicked on)

Component programmers should call `Execute` whenever the action should be activated.

See also: Action (345), TBasicAction.ActionComponent (342), TBasicAction.Execute (341), TBasicAction.onExecute (342)

4.38.7 TBasicActionLink.Update

Synopsis: Calls the action's Update method.

Declaration: `function Update : Boolean;  Virtual`

Visibility: public

Description: `Update` calls the associated Action's Update (342) method.

Component programmers can override the `Update` method to provide additional processing when the `Update` method occurs.

4.38.8 TBasicActionLink.Action

Synopsis: The action to which the link was assigned.

Declaration: `Property Action : TBasicAction`

Visibility: public

Access: Read,Write

Description: `Action` represents the Action (339) which was assigned to the client. Setting this property will unregister the client at the old action (if one existed) and registers the client at the new action.

See also: TBasicAction (339)

4.38.9 TBasicActionLink.OnChange

Synopsis: Event handler triggered when the action's properties change.

Declaration: `Property OnChange : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnChange` is the event triggered when the action's properties change.

Application programmers should never need to assign this event. Component programmers can assign this event to have a client control reflect any changes in an Action's properties.

See also: Change (343), TBasicAction.Change (339)

4.39 TBinaryObjectReader

4.39.1 Description

The `TBinaryObjectReader` class reads component data stored in binary form in a file. For this, it overrides or implements all abstract methods from TAbstractObjectReader (325). No new functionality is added by this class, it is a driver class for the streaming system.

It should never be necessary to create an instance of this class directly. Instead, the TStream.WriteComponent (460) call should be used.

See also: TAbstractObjectReader (325), TBinaryObjectWriter (352)

4.39.2 Method overview

Page	Method	Description
347	BeginComponent	Start reading a component.
348	BeginProperty	Start reading a property.
347	BeginRootComponent	Start reading the root component.
346	Create	Creates a new binary data reader instance.
346	Destroy	Destroys the binary data reader.
347	NextValue	Return the type of the next value.
348	Read	Read raw data from stream.
348	ReadBinary	Start reading a binary value.
349	ReadCurrency	Read a currency value from the stream.
349	ReadDate	Read a date.
348	ReadFloat	Read a float value.
349	ReadIdent	Read an identifier.
349	ReadInt16	Read a 16-bits integer.
350	ReadInt32	Read a 32-bits integer.
350	ReadInt64	Read a 64-bits integer.
349	ReadInt8	Read an 8-bits integer.
350	ReadSet	Read a set.
350	ReadSignature	Reads the filer signature.
348	ReadSingle	Read a single-size float value.
351	ReadStr	Read a short string.
351	ReadString	Read a string.
351	ReadUnicodeString	Read a Unicode string value.
347	ReadValue	Read the next value in the stream.
351	ReadWideString	Read a widestring value from the stream.
352	SkipComponent	Skip a component's data.
352	SkipValue	Skip a value's data.

4.39.3 TBinaryObjectReader.Create

Synopsis: Creates a new binary data reader instance.

Declaration: `constructor Create(Stream: TStream; BufSize: Integer)`

Visibility: public

Description: `Create` instantiates a new binary component data reader. The `Stream` stream is the stream from which data will be read. The `BufSize` argument is the size of the internal buffer that will be used by the reader. This can be used to optimize the reading process.

See also: TAbstractObjectReader (325)

4.39.4 TBinaryObjectReader.Destroy

Synopsis: Destroys the binary data reader.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` frees the buffer allocated when the instance was created. It also positions the stream on the last used position in the stream (the buffering may cause the reader to read more bytes than were actually used.)

See also: TBinaryObjectReader.Create (346)

4.39.5 TBinaryObjectReader.NextValue

Synopsis: Return the type of the next value.

Declaration:
```
function NextValue : TValueType;  Override
```

Visibility: public

Description: `NextValue` returns the type of the next value in a binary stream, but does not read the value.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (325)

4.39.6 TBinaryObjectReader.ReadValue

Synopsis: Read the next value in the stream.

Declaration:
```
function ReadValue : TValueType;  Override
```

Visibility: public

Description: `NextValue` reads the next value in a binary stream and returns the type of the read value.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (325)

4.39.7 TBinaryObjectReader.BeginRootComponent

Synopsis: Start reading the root component.

Declaration:
```
procedure BeginRootComponent;  Override
```

Visibility: public

Description: `BeginRootComponent` starts reading the root component in a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (325)

4.39.8 TBinaryObjectReader.BeginComponent

Synopsis: Start reading a component.

Declaration:
```
procedure BeginComponent(var Flags: TFilerFlags;
                         var AChildPos: Integer;
                         var CompClassName: string; var CompName: string)
                         ;  Override
```

Visibility: public

Description: This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (325)

4.39.9 TBinaryObjectReader.BeginProperty

Synopsis: Start reading a property.

Declaration: `function BeginProperty : string;  Override`

Visibility: public

Description: This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (325)

4.39.10 TBinaryObjectReader.Read

Synopsis: Read raw data from stream.

Declaration: `procedure Read(var Buf; Count: LongInt);  Override`

Visibility: public

Description: `Read` is introduced for Delphi compatibility to read raw data from the component stream. This should not be used in production code as it will totally mess up the streaming.

See also: TAbstractObjectReader.Read (328), TReader.Read (444)

4.39.11 TBinaryObjectReader.ReadBinary

Synopsis: Start reading a binary value.

Declaration: `procedure ReadBinary(const DestData: TMemoryStream);  Override`

Visibility: public

Description: `ReadBinary` reads a binary value from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (325)

4.39.12 TBinaryObjectReader.ReadFloat

Synopsis: Read a float value.

Declaration: `function ReadFloat : Extended;  Override`

Visibility: public

Description: `ReadFloat` reads a float value from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (325)

4.39.13 TBinaryObjectReader.ReadSingle

Synopsis: Read a single-size float value.

Declaration: `function ReadSingle : Single;  Override`

Visibility: public

Description: `ReadSingle` reads a single-sized float value from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (325)

4.39.14 TBinaryObjectReader.ReadDate

Synopsis: Read a date.

Declaration: `function ReadDate : TDateTime;  Override`

Visibility: public

Description: `ReadDate` reads a date value from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (325)

4.39.15 TBinaryObjectReader.ReadCurrency

Synopsis: Read a currency value from the stream.

Declaration: `function ReadCurrency : Currency;  Override`

Visibility: public

Description: `var>ReadCurrency` reads a currency-typed value from a binary stream. It is the implementation of the method introduced in TAbstractObjectReader (325).

See also: TAbstractObjectReader.ReadCurrency (329), TBinaryObjectWriter.WriteCurrency (356)

4.39.16 TBinaryObjectReader.ReadIdent

Synopsis: Read an identifier.

Declaration: `function ReadIdent(ValueType: TValueType) : string;  Override`

Visibility: public

Description: `ReadIdent` reads an identifier from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (325)

4.39.17 TBinaryObjectReader.ReadInt8

Synopsis: Read an 8-bits integer.

Declaration: `function ReadInt8 : ShortInt;  Override`

Visibility: public

Description: `Read8Int` reads an 8-bits signed integer from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (325)

4.39.18 TBinaryObjectReader.ReadInt16

Synopsis: Read a 16-bits integer.

Declaration: `function ReadInt16 : SmallInt;  Override`

Visibility: public

Description: `Read16Int` reads a 16-bits signed integer from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (325)

4.39.19 TBinaryObjectReader.ReadInt32

Synopsis: Read a 32-bits integer.

Declaration: `function ReadInt32 : LongInt;  Override`

Visibility: public

Description: `Read32Int` reads a 32-bits signed integer from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (325)

4.39.20 TBinaryObjectReader.ReadInt64

Synopsis: Read a 64-bits integer.

Declaration: `function ReadInt64 : Int64;  Override`

Visibility: public

Description: `Read64Int` reads a 64-bits signed integer from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (325)

4.39.21 TBinaryObjectReader.ReadSet

Synopsis: Read a set.

Declaration: `function ReadSet(EnumType: Pointer) : Integer;  Override`

Visibility: public

Description: `ReadSet` reads a set from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (325)

4.39.22 TBinaryObjectReader.ReadSignature

Synopsis: Reads the filer signature.

Declaration: `procedure ReadSignature;  Override`

Visibility: public

Description: `ReadSignature` is overridden by `TBinaryObjectReader` to read the signature (`TPF0`) from binary streams.

Errors: If the stream does not start with the correct signature, an EReadError (311) exception is raised.

See also: TAbstractObjectReader.ReadSignature (332), TAbstractObjectWriter.WriteSignature (334), TBinaryObjectWriter.WriteSignature (353)

4.39.23 TBinaryObjectReader.ReadStr

Synopsis: Read a short string.

Declaration: `function ReadStr : string;  Override`

Visibility: public

Description: `ReadStr` reads a short string from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (325)

4.39.24 TBinaryObjectReader.ReadString

Synopsis: Read a string.

Declaration: `function ReadString(StringType: TValueType) : string;  Override`

Visibility: public

Description: `ReadStr` reads a string of type `StringType` from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (325)

4.39.25 TBinaryObjectReader.ReadWideString

Synopsis: Read a widestring value from the stream.

Declaration: `function ReadWideString : WideString;  Override`

Visibility: public

Description: `var>ReadWideString` reads a widestring-typed value from a binary stream. It is the implementation of the method introduced in TAbstractObjectReader (325).

See also: TAbstractObjectReader.ReadWideString (332), TBinaryObjectWriter.WriteWideString (357)

4.39.26 TBinaryObjectReader.ReadUnicodeString

Synopsis: Read a Unicode string value.

Declaration: `function ReadUnicodeString : UnicodeString;  Override`

Visibility: public

Description: `ReadUnicodeString` is overridden by `TBinaryObjectReader`to read a UnicodeString value from the binary stream.

See also: TAbstractObjectReader.ReadUnicodeString (333)

4.39.27 TBinaryObjectReader.SkipComponent

Synopsis: Skip a component's data.

Declaration: `procedure SkipComponent(SkipComponentInfos: Boolean); Override`

Visibility: public

Description: `SkipComponent` skips the data of a component in a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (325).

4.39.28 TBinaryObjectReader.SkipValue

Synopsis: Skip a value's data.

Declaration: `procedure SkipValue; Override`

Visibility: public

Description: `SkipComponent` skips the data of the next value in a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader (325)

4.40 TBinaryObjectWriter

4.40.1 Description

Driver class which stores component data in binary form.

4.40.2 Method overview

Page	Method	Description
354	BeginCollection	Start writing a collection.
354	BeginComponent	Start writing a component.
354	BeginList	Start writing a list.
355	BeginProperty	Start writing a property.
353	Create	Creates a new instance of a binary object writer.
353	Destroy	Destroys an instance of the binary object writer.
354	EndList	Mark the end of a list.
355	EndProperty	Marks the end of writing of a property.
354	FlushBuffer	Flush the buffer
355	Write	Write raw data to stream.
355	WriteBinary	Writes binary data to the stream.
355	WriteBoolean	Writes a boolean value to the stream.
356	WriteCurrency	Write a currency-valued type to a stream.
356	WriteDate	Writes a date type to the stream.
355	WriteFloat	Writes a float value to the stream.
356	WriteIdent	Writes an identifier to the stream.
356	WriteInteger	Writes an integer value to the stream.
357	WriteMethodName	Writes a methodname to the stream.
357	WriteSet	Writes a set value to the stream.
353	WriteSignature	Write stream signature to the stream.
356	WriteSingle	Writes a single-type real value to the stream.
357	WriteStr	Write a string to the binary stream.
357	WriteString	Writes a string value to the stream.
356	WriteUInt64	Write an unsigned 64-bit integer.
357	WriteUnicodeString	Write a Unicode string to the stream.
358	WriteVariant	Write a variant to the stream.
357	WriteWideString	Write a widestring-valued type to a stream.

4.40.3 TBinaryObjectWriter.Create

Synopsis: Creates a new instance of a binary object writer.

Declaration: `constructor Create(Stream: TStream; BufSize: Integer)`

Visibility: public

Description: Creates a new instance of a binary object writer.

4.40.4 TBinaryObjectWriter.Destroy

Synopsis: Destroys an instance of the binary object writer.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Destroys an instance of the binary object writer.

4.40.5 TBinaryObjectWriter.WriteSignature

Synopsis: Write stream signature to the stream.

Declaration: `procedure WriteSignature;  Override`

Visibility: public

Description: `WriteSignature` is overridden by `TBinaryObjectWriter` to write the signature (`TPF0`) to binary streams.

See also: TAbstractObjectWriter.WriteSignature (334), TAbstractObjectReader.ReadSignature (332), TBinaryObjectReader.ReadSignature (350)

4.40.6 TBinaryObjectWriter.FlushBuffer

Synopsis: Flush the buffer

Declaration: `procedure FlushBuffer;  Override`

Visibility: public

Description: `FlushBuffer` flushes the buffer. It is provided for Delphi compatibility, and is not used in FPC.

See also: TFiler.FlushBuffer (394), TWriter.FlushBuffer (526), TAbstractObjectWriter.FlushBuffer (335)

4.40.7 TBinaryObjectWriter.BeginCollection

Synopsis: Start writing a collection.

Declaration: `procedure BeginCollection;  Override`

Visibility: public

4.40.8 TBinaryObjectWriter.BeginComponent

Synopsis: Start writing a component.

Declaration:
```
procedure BeginComponent(Component: TComponent; Flags: TFilerFlags;
                         ChildPos: Integer);  Override
```

Visibility: public

4.40.9 TBinaryObjectWriter.BeginList

Synopsis: Start writing a list.

Declaration: `procedure BeginList;  Override`

Visibility: public

4.40.10 TBinaryObjectWriter.EndList

Synopsis: Mark the end of a list.

Declaration: `procedure EndList;  Override`

Visibility: public

4.40.11 TBinaryObjectWriter.BeginProperty

Synopsis: Start writing a property.

Declaration: `procedure BeginProperty(const PropName: string);  Override`

Visibility: public

4.40.12 TBinaryObjectWriter.EndProperty

Synopsis: Marks the end of writing of a property.

Declaration: `procedure EndProperty;  Override`

Visibility: public

4.40.13 TBinaryObjectWriter.Write

Synopsis: Write raw data to stream.

Declaration: `procedure Write(const Buffer; Count: LongInt);  Override`

Visibility: public

Description: `Write` is introduced for Delphi compatibility to write raw data to the component stream. This should not be used in new production code as it will totally mess up the streaming.

See also: TAbstractObjectWriter.Write (336), TWriter.Write (526)

4.40.14 TBinaryObjectWriter.WriteBinary

Synopsis: Writes binary data to the stream.

Declaration: `procedure WriteBinary(const Buffer; Count: LongInt);  Override`

Visibility: public

4.40.15 TBinaryObjectWriter.WriteBoolean

Synopsis: Writes a boolean value to the stream.

Declaration: `procedure WriteBoolean(Value: Boolean);  Override`

Visibility: public

4.40.16 TBinaryObjectWriter.WriteFloat

Synopsis: Writes a float value to the stream.

Declaration: `procedure WriteFloat(const Value: Extended);  Override`

Visibility: public

4.40.17 TBinaryObjectWriter.WriteSingle

Synopsis: Writes a single-type real value to the stream.

Declaration: `procedure WriteSingle(const Value: Single);  Override`

Visibility: public

4.40.18 TBinaryObjectWriter.WriteDate

Synopsis: Writes a date type to the stream.

Declaration: `procedure WriteDate(const Value: TDateTime);  Override`

Visibility: public

4.40.19 TBinaryObjectWriter.WriteCurrency

Synopsis: Write a currency-valued type to a stream.

Declaration: `procedure WriteCurrency(const Value: Currency);  Override`

Visibility: public

Description: `WriteCurrency` writes a currency-typed value to a binary stream. It is the implementation of the method introduced in TAbstractObjectWriter (333).

See also: TAbstractObjectWriter.WriteCurrency (337)

4.40.20 TBinaryObjectWriter.WriteIdent

Synopsis: Writes an identifier to the stream.

Declaration: `procedure WriteIdent(const Ident: string);  Override`

Visibility: public

4.40.21 TBinaryObjectWriter.WriteInteger

Synopsis: Writes an integer value to the stream.

Declaration: `procedure WriteInteger(Value: Int64);  Override`

Visibility: public

4.40.22 TBinaryObjectWriter.WriteUInt64

Synopsis: Write an unsigned 64-bit integer.

Declaration: `procedure WriteUInt64(Value: QWord);  Override`

Visibility: public

Description: `WriteUInt64` is overridden by `TBinaryObjectWriter` to write an unsigned 64-bit integer (QWord) to the stream. It tries to use the smallest possible storage for the value that is passed. (largest valuetype will be `vaQWord`).

See also: TAbstractObjectWriter.WriteUInt64 (337)

4.40.23 TBinaryObjectWriter.WriteMethodName

Synopsis: Writes a methodname to the stream.

Declaration: `procedure WriteMethodName(const Name: string);  Override`

Visibility: public

4.40.24 TBinaryObjectWriter.WriteSet

Synopsis: Writes a set value to the stream.

Declaration: `procedure WriteSet(Value: LongInt; SetType: Pointer);  Override`

Visibility: public

4.40.25 TBinaryObjectWriter.WriteStr

Synopsis: Write a string to the binary stream.

Declaration: `procedure WriteStr(const Value: string)`

Visibility: public

Description: `WriteStr` writes a string value to the binary stream. It is exposed so it can be used in `DefineProperties`.

4.40.26 TBinaryObjectWriter.WriteString

Synopsis: Writes a string value to the stream.

Declaration: `procedure WriteString(const Value: string);  Override`

Visibility: public

4.40.27 TBinaryObjectWriter.WriteWideString

Synopsis: Write a widestring-valued type to a stream.

Declaration: `procedure WriteWideString(const Value: WideString);  Override`

Visibility: public

Description: `WriteWidestring` writes a widestring-typed value to a binary stream. It is the implementation of the method introduced in TAbstractObjectWriter (333).

See also: TAbstractObjectWriter.WriteWidestring (338)

4.40.28 TBinaryObjectWriter.WriteUnicodeString

Synopsis: Write a Unicode string to the stream.

Declaration: `procedure WriteUnicodeString(const Value: UnicodeString);  Override`

Visibility: public

Description: `WriteUnicodeString` is overridden `TBinaryObjectWriter` to write a unicodestring (valuetype `vaUString`) value to the stream. It simply writes the character length and then all widecharacters.

See also: TAbstractObjectWriter.WriteUnicodeString (339)

4.40.29 TBinaryObjectWriter.WriteVariant

Synopsis: Write a variant to the stream.

Declaration: `procedure WriteVariant(const VarValue: Variant);  Override`

Visibility: public

Description: `WriteVariant` is overridden by `TBinaryObjectWriter` to write a simple variant type to the stream. `WriteVariant` does not write arrays types or complex types. Only null, integer (ordinal) float and string types are written.

Errors: If a non-supported type is written, then an `EWriteError` exception is.

4.41 TBits

4.41.1 Description

`TBits` can be used to store collections of bits in an indexed array. This is especially useful for storing collections of booleans: Normally the size of a boolean is the size of the smallest enumerated type, i.e. 1 byte. Since a bit can take 2 values it can be used to store a boolean as well. Since TBits can store 8 bits in a byte, it takes 8 times less space to store an array of booleans in a TBits class then it would take to store them in a conventional array.

`TBits` introduces methods to store and retrieve bit values, apply masks, and search for bits.

4.41.2 Method overview

Page	Method	Description
360	AndBits	Performs an `and` operation on the bits.
360	Clear	Clears a particular bit.
360	Clearall	Clears all bits in the array.
360	CopyBits	Copy bits from one bits set to another.
359	Create	Creates a new bits collection.
359	Destroy	Destroys a bit collection.
362	Equals	Determines whether the bits of 2 arrays are equal.
363	FindFirstBit	Find first bit with a particular value.
363	FindNextBit	Searches the next bit with a particular value.
363	FindPrevBit	Searches the previous bit with a particular value.
362	Get	Retrieve the value of a particular bit.
359	GetFSize	Returns the number of records used to store the bits.
362	Grow	Expands the bits array to the requested size.
361	NotBits	Performs a `not` operation on the bits.
364	OpenBit	Returns the position of the first bit that is set to `False`.
361	OrBits	Performs an `or` operation on the bits.
362	SetIndex	Sets the start position for FindNextBit (363) and FindPrevBit (363).
359	SetOn	Turn a particular bit on.
361	XorBits	Performs a `xor` operation on the bits.

4.41.3 Property overview

Page	Properties	Access	Description
364	Bits	rw	Access to all bits in the array.
364	Size	rw	Current size of the array of bits.

4.41.4 TBits.Create

Synopsis: Creates a new bits collection.

Declaration: `constructor Create(TheSize: LongInt); Virtual`

Visibility: public

Description: `Create` creates a new bit collection with initial size `TheSize`. The size of the collection can be changed later on.

All bits are initially set to zero.

See also: Destroy (359)

4.41.5 TBits.Destroy

Synopsis: Destroys a bit collection.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` destroys a previously created bit collection and releases all memory used to store the bit collection.

`Destroy` should never be called directly, `Free` should be used instead.

Errors: None.

See also: Create (359)

4.41.6 TBits.GetFSize

Synopsis: Returns the number of records used to store the bits.

Declaration: `function GetFSize : LongInt`

Visibility: public

Description: `GetFSize` returns the number of records used to store the current number of bits.

Errors: None.

See also: Size (364)

4.41.7 TBits.SetOn

Synopsis: Turn a particular bit on.

Declaration: `procedure SetOn(Bit: LongInt)`

Visibility: public

Description: `SetOn` turns on the bit at position `bit`, i.e. sets it to 1. If `bit` is at a position bigger than the current size, the collection is expanded to the required size using Grow (362).

Errors: If `bit` is larger than the maximum allowed bits array size or is negative, an EBitsError (309) exception is raised.

See also: Bits (364), Clear (360)

4.41.8 TBits.Clear

Synopsis: Clears a particular bit.

Declaration: `procedure Clear(Bit: LongInt)`

Visibility: public

Description: `Clear` clears the bit at position `bit`. If the array If `bit` is at a position bigger than the current size, the collection is expanded to the required size using Grow (362).

Errors: If `bit` is larger than the maximum allowed bits array size or is negative, an EBitsError (309) exception is raised.

See also: Bits (364), seton (360)

4.41.9 TBits.Clearall

Synopsis: Clears all bits in the array.

Declaration: `procedure Clearall`

Visibility: public

Description: `ClearAll` clears all bits in the array, i.e. sets them to zero. `ClearAll` works faster than clearing all individual bits, since it uses the packed nature of the bits.

Errors: None.

See also: Bits (364), clear (360)

4.41.10 TBits.CopyBits

Synopsis: Copy bits from one bits set to another.

Declaration: `procedure CopyBits(BitSet: TBits)`

Visibility: public

Description: `CopyBits` copies the bits from `BitSet` to the current Bits. Existing bits will be overwritten. The two sets of bits will be equal after this operation.

Errors: None.

See also: TBits.Equals (362)

4.41.11 TBits.AndBits

Synopsis: Performs an `and` operation on the bits.

Declaration: `procedure AndBits(BitSet: TBits)`

Visibility: public

Description: `andbits` performs an `and` operation on the bits in the array with the bits of array `BitSet`. If `BitSet` contains less bits than the current array, then all bits which have no counterpart in `BitSet` are cleared.

Errors: None.

See also: ClearAll (360), OrBits (361), XOrBits (361), NotBits (361)

4.41.12 TBits.OrBits

Synopsis: Performs an `or` operation on the bits.

Declaration: `procedure OrBits(BitSet: TBits)`

Visibility: public

Description: `andbits` performs an `or` operation on the bits in the array with the bits of array `BitSet`.

If `BitSet` contains less bits than the current array, then all bits which have no counterpart in `BitSet` are left untouched.

If the current array contains less bits than `BitSet` then it is grown to the size of `BitSet` before the `or` operation is performed.

Errors: None.

See also: ClearAll (360), andBits (360), XOrBits (361), NotBits (361)

4.41.13 TBits.XorBits

Synopsis: Performs a `xor` operation on the bits.

Declaration: `procedure XorBits(BitSet: TBits)`

Visibility: public

Description: `XorBits` performs a `xor` operation on the bits in the array with the bits of array `BitSet`.

If `BitSet` contains less bits than the current array, then all bits which have no counterpart in `BitSet` are left untouched.

If the current array contains less bits than `BitSet` then it is grown to the size of `BitSet` before the `xor` operation is performed.

Errors: None.

See also: ClearAll (360), andBits (360), OrBits (361), NotBits (361)

4.41.14 TBits.NotBits

Synopsis: Performs a `not` operation on the bits.

Declaration: `procedure NotBits(BitSet: TBits)`

Visibility: public

Description: `NotBits` performs a `not` operation on the bits in the array with the bits of array `Bitset`.

If `BitSet` contains less bits than the current array, then all bits which have no counterpart in `BitSet` are left untouched.

Errors: None.

See also: ClearAll (360), andBits (360), OrBits (361), XOrBits (361)

4.41.15 TBits.Get

Synopsis: Retrieve the value of a particular bit.

Declaration:
```
function Get(Bit: LongInt) : Boolean
```

Visibility: public

Description: `Get` returns `True` if the bit at position `bit` is set, or `False` if it is not set.

Errors: If `bit` is not a valid bit index then an EBitsError (309) exception is raised.

See also: Bits (364), FindFirstBit (363), seton (359)

4.41.16 TBits.Grow

Synopsis: Expands the bits array to the requested size.

Declaration:
```
procedure Grow(NBit: LongInt)
```

Visibility: public

Description: `Grow` expands the bit array so it can at least contain `nbit` bits. If `nbit` is less than the current size, nothing happens.

Errors: If there is not enough memory to complete the operation, then an EBitsError (309)is raised.

See also: Size (364)

4.41.17 TBits.Equals

Synopsis: Determines whether the bits of 2 arrays are equal.

Declaration:
```
function Equals(Obj: TObject) : Boolean;  Override;  Overload
function Equals(BitSet: TBits) : Boolean;  Overload
```

Visibility: public

Description: `equals` returns `True` if all the bits in `BitSet` are the same as the ones in the current BitSet; if not, `False` is returned.

If the sizes of the two BitSets are different, the arrays are still reported equal when all the bits in the larger set, which are not present in the smaller set, are zero.

Errors: None.

See also: ClearAll (360), andBits (360), OrBits (361), XOrBits (361)

4.41.18 TBits.SetIndex

Synopsis: Sets the start position for FindNextBit (363) and FindPrevBit (363).

Declaration:
```
procedure SetIndex(Index: LongInt)
```

Visibility: public

Description: `SetIndex` sets the search start position forFindNextBit (363) and FindPrevBit (363) to `Index`. This means that these calls will start searching from position `Index`.

This mechanism provides an alternative to FindFirstBit (363) which can also be used to position for the `FindNextBit` and `FindPrevBit` calls.

Errors: None.

See also: FindNextBit (363), FindPrevBit (363), FindFirstBit (363), OpenBit (364)

4.41.19 TBits.FindFirstBit

Synopsis: Find first bit with a particular value.

Declaration: `function FindFirstBit(State: Boolean) : LongInt`

Visibility: public

Description: `FindFirstBit` searches for the first bit with value `State`. It returns the position of this bit, or `-1` if no such bit was found.

The search starts at position 0 in the array. If the first search returned a positive result, the found position is saved, and the FindNextBit (363) and FindPrevBit (363) will use this position to resume the search. To start a search from a certain position, the start position can be set with the SetIndex (362) instead.

Errors: None.

See also: FindNextBit (363), FindPrevBit (363), OpenBit (364), SetIndex (362)

4.41.20 TBits.FindNextBit

Synopsis: Searches the next bit with a particular value.

Declaration: `function FindNextBit : LongInt`

Visibility: public

Description: `FindNextBit` resumes a previously started search. It searches for the next bit with the value specified in the FindFirstBit (363). The search is done towards the end of the array and starts at the position last reported by one of the `Find` calls or at the position set with SetIndex (362).

If another bit with the same value is found, its position is returned. If no more bits with the same value are present in the array, `-1` is returned.

Errors: None.

See also: FindFirstBit (363), FindPrevBit (363), OpenBit (364), SetIndex (362)

4.41.21 TBits.FindPrevBit

Synopsis: Searches the previous bit with a particular value.

Declaration: `function FindPrevBit : LongInt`

Visibility: public

Description: `FindPrevBit` resumes a previously started search. It searches for the previous bit with the value specified in the FindFirstBit (363). The search is done towards the beginning of the array and starts at the position last reported by one of the `Find` calls or at the position set with SetIndex (362).

If another bit with the same value is found, its position is returned. If no more bits with the same value are present in the array, `-1` is returned.

Errors: None.

See also: FindFirstBit (363), FindNextBit (363), OpenBit (364), SetIndex (362)

4.41.22 TBits.OpenBit

Synopsis: Returns the position of the first bit that is set to `False`.

Declaration: `function OpenBit : LongInt`

Visibility: public

Description: `OpenBit` returns the position of the first bit whose value is `0` (`False`), or `-1` if no open bit was found. This call is equivalent to `FindFirstBit(False)`, except that it doesn't set the position for the next searches.

Errors: None.

See also: FindNextBit (363), FindPrevBit (363), FindFirstBit (363), SetIndex (362)

4.41.23 TBits.Bits

Synopsis: Access to all bits in the array.

Declaration: `Property Bits[Bit: LongInt]: Boolean; default`

Visibility: public

Access: Read,Write

Description: `Bits` allows indexed access to all of the bits in the array. It gives `True` if the bit is `1`, `False` otherwise; Assigning to this property will set, respectively clear the bit.

Errors: If an index is specified which is out of the allowed range then an EBitsError (309) exception is raised.

See also: Size (364)

4.41.24 TBits.Size

Synopsis: Current size of the array of bits.

Declaration: `Property Size : LongInt`

Visibility: public

Access: Read,Write

Description: `Size` is the current size of the bit array. Setting this property will adjust the size; this is equivalent to calling `Grow(Value-1)`

Errors: If an invalid size (negative or too large) is specified, a EBitsError (309) exception is raised.

See also: Bits (364)

4.42 TBytesStream

4.42.1 Description

`TBytesStream` is a stream that uses an array of byte (TBytes (272)) to keep the stream data. it overrides the TMemoryStream (425) memory allocation routine to use the array of bytes. The array of bytes is exposed through the Bytes (272) property.

See also: TBytes (272), TMemoryStream (425), Bytes (272)

4.42.2 Method overview

Page	Method	Description
365	Create	Create a new instance of the stream, initializing it with an array of bytes.

4.42.3 Property overview

Page	Properties	Access	Description
365	Bytes	r	The stream data as an array of bytes.

4.42.4 TBytesStream.Create

Synopsis: Create a new instance of the stream, initializing it with an array of bytes.

Declaration: `constructor Create(const ABytes: TBytes); Virtual; Overload`

Visibility: public

Description: `Create` creates a new instance and initializes the memory with the data in `ABytes`.

See also: TBytes (272), TMemoryStream (425), Bytes (272)

4.42.5 TBytesStream.Bytes

Synopsis: The stream data as an array of bytes.

Declaration: `Property Bytes : TBytes`

Visibility: public

Access: Read

Description: `Bytes` provides byte-sized access to the array of bytes that represent the stream data. As a pointer value, it equals TCustomMemoryStream.Memory (389), meaning that `Memory` points to the first byte in the array.

See also: TBytes (272), TMemoryStream (425), TCustomMemoryStream.Memory (389)

4.43 TCollection

4.43.1 Description

`TCollection` implements functionality to manage a collection of named objects. Each of these objects needs to be a descendant of the TCollectionItem (373) class. Exactly which type of object is managed can be seen from the TCollection.ItemClass (371) property.

Normally, no `TCollection` is created directly. Instead, a descendants of `TCollection` and TCollectionItem (373) are created as a pair.

See also: TCollectionItem (373)

4.43.2 Method overview

Page	Method	Description
367	Add	Creates and adds a new item to the collection.
367	Assign	Assigns one collection to another.
367	BeginUpdate	Start an update batch.
368	Clear	Removes all items from the collection.
366	Create	Creates a new collection.
368	Delete	Delete an item from the collection.
366	Destroy	Destroys the collection and frees all the objects it manages.
368	EndUpdate	Ends an update batch.
370	Exchange	Exchange 2 items in the collection.
369	FindItemID	Searches for an Item in the collection, based on its TCollectionItem.ID (375) property.
369	GetEnumerator	Create an `IEnumerator` instance.
369	GetNamePath	Overrides TPersistent.GetNamePath (437) to return a proper pathname.
369	Insert	Insert an item in the collection.
370	Move	Move an item in the collection.
367	Owner	Owner of the collection.
370	Sort	Sort the items in the collection.

4.43.3 Property overview

Page	Properties	Access	Description
371	Count	r	Number of items in the collection.
371	ItemClass	r	Class pointer for each item in the collection.
371	Items	rw	Indexed array of items in the collection.

4.43.4 TCollection.Create

Synopsis: Creates a new collection.

Declaration: `constructor Create(AItemClass: TCollectionItemClass)`

Visibility: public

Description: `Create` instantiates a new instance of the `TCollection` class which will manage objects of class `AItemClass`. It creates the list used to hold all objects, and stores the `AItemClass` for the adding of new objects to the collection.

See also: TCollection.ItemClass (371), TCollection.Destroy (366)

4.43.5 TCollection.Destroy

Synopsis: Destroys the collection and frees all the objects it manages.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` first clears the collection, and then frees all memory allocated to this instance.

Don't call `Destroy` directly, call `Free` instead.

See also: TCollection.Create (366)

4.43.6 TCollection.Owner

Synopsis: Owner of the collection.

Declaration: `function Owner : TPersistent`

Visibility: public

Description: `Owner` returns a reference to the owner of the collection. This property is required by the object inspector to be able to show the collection.

4.43.7 TCollection.Add

Synopsis: Creates and adds a new item to the collection.

Declaration: `function Add : TCollectionItem`

Visibility: public

Description: `Add` instantiates a new item of class TCollection.ItemClass (371) and adds it to the list. The newly created object is returned.

See also: TCollection.ItemClass (371), TCollection.Clear (368)

4.43.8 TCollection.Assign

Synopsis: Assigns one collection to another.

Declaration: `procedure Assign(Source: TPersistent);  Override`

Visibility: public

Description: `Assign` assigns the contents of one collection to another. It does this by clearing the items list, and adding as much elements as there are in the `Source` collection; it assigns to each created element the contents of it's counterpart in the `Source` element.

Two collections cannot be assigned to each other if instances of the `ItemClass` classes cannot be assigned to each other.

Errors: If the objects in the collections cannot be assigned to one another, then an `EConvertError` is raised.

See also: TPersistent.Assign (436), TCollectionItem (373)

4.43.9 TCollection.BeginUpdate

Synopsis: Start an update batch.

Declaration: `procedure BeginUpdate;  Virtual`

Visibility: public

Description: `BeginUpdate` is called at the beginning of a batch update. It raises the update count with 1.

Call `BeginUpdate` at the beginning of a series of operations that will change the state of the collection. This will avoid the call to TCollection.Update (365) for each operation. At the end of the operations, a corresponding call to `EndUpdate` must be made. It is best to do this in the context of a `Try ... finally` block:

```
With MyCollection Do
    try
      BeginUpdate;
      // Some Lengthy operations
    finally
      EndUpdate;
    end;
```

This insures that the number of calls to `BeginUpdate` always matches the number of calls to TCollection.EndUpdate (368), even in case of an exception.

See also: TCollection.EndUpdate (368), TCollection.Changed (365), TCollection.Update (365)

4.43.10 TCollection.Clear

Synopsis: Removes all items from the collection.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` will clear the collection, i.e. each item in the collection is destroyed and removed from memory. After a call to `Clear`, `Count` is zero.

See also: TCollection.Add (367), TCollectionItem.Destroy (374), TCollection.Destroy (366)

4.43.11 TCollection.EndUpdate

Synopsis: Ends an update batch.

Declaration: `procedure EndUpdate;  Virtual`

Visibility: public

Description: `EndUpdate` signals the end of a series of operations that change the state of the collection, possibly triggering an update event. It does this by decreasing the update count with 1 and calling TCollection.Changed (365) it should always be used in conjunction with TCollection.BeginUpdate (367), preferably in the `Finally` section of a `Try ... Finally` block.

See also: TCollection.BeginUpdate (367), TCollection.Changed (365), TCollection.Update (365)

4.43.12 TCollection.Delete

Synopsis: Delete an item from the collection.

Declaration: `procedure Delete(Index: Integer)`

Visibility: public

Description: `Delete` deletes the item at (zero based) position `Index` from the collection. This will result in a `cnDeleted` notification.

Errors: If an invalid index is specified, an `EListError` exception is raised.

See also: TCollection.Items (371), TCollection.Insert (369), TCollection.Clear (368)

4.43.13 TCollection.GetEnumerator

Synopsis: Create an `IEnumerator` instance.

Declaration: `function GetEnumerator : TCollectionEnumerator`

Visibility: public

Description: `GetEnumerator` is the implementation of the IEnumerable (1616) interface for `TCollection`. It creates a TCollectionEnumerator (372) instance and returns it's IEnumerator (1617) interface.

See also: TCollectionEnumerator (372), IEnumerator (1617), IEnumerable (1616)

4.43.14 TCollection.GetNamePath

Synopsis: Overrides TPersistent.GetNamePath (437) to return a proper pathname.

Declaration: `function GetNamePath : string;  Override`

Visibility: public

Description: `GetNamePath` returns the name path for this collection. If the following conditions are satisfied:

1.There is an owner object.

2.The owner object returns a non-empty name path.

3.The TCollection.Propname (365) property is not empty

collection has an owner and the owning object has a name, then the function returns the owner name, followed by the propname. If one of the conditions is not satisfied, then the classname is returned.

See also: TCollection.GetOwner (365), TCollection.Propname (365)

4.43.15 TCollection.Insert

Synopsis: Insert an item in the collection.

Declaration: `function Insert(Index: Integer) : TCollectionItem`

Visibility: public

Description: `Insert` creates a new item instance and inserts it in the collection at position `Index`, and returns the new instance.

In contrast, TCollection.Add (367) adds a new item at the end.

Errors: None.

See also: TCollection.Add (367), TCollection.Delete (368), TCollection.Items (371)

4.43.16 TCollection.FindItemID

Synopsis: Searches for an Item in the collection, based on its TCollectionItem.ID (375) property.

Declaration: `function FindItemID(ID: Integer) : TCollectionItem`

Visibility: public

Description: `FindItemID` searches through the collection for the item that has a value of `ID` for its TCollectionItem.ID (375) property, and returns the found item. If no such item is found in the collection, `Nil` is returned.

The routine performs a linear search, so this can be slow on very large collections.

See also: TCollection.Items (371), TCollectionItem.ID (375)

4.43.17 TCollection.Exchange

Synopsis: Exchange 2 items in the collection.

Declaration: `procedure Exchange(const Index1: Integer; const index2: Integer)`

Visibility: public

Description: `Exchange` exchanges the items at indexes `Index1` and `Index2` in the collection.

Errors: If one of the two indexes is invalid (less than zero or larger than the number of items) an `EListError` exception is raised.

See also: Items (371), TCollectionItem.Index (375)

4.43.18 TCollection.Move

Synopsis: Move an item in the collection.

Declaration: `procedure Move(const Index1: Integer; const index2: Integer)`

Visibility: public

Description: `Move` will move the collection item from position `index1` to `Index2`. In difference with Exchange (370) the indexes of the elements between `index1` to `Index2` will shift.

Errors: If an invalid index is specified in `index1` or `Index2`, an exception will be raised.

See also: TCollection.Exchange (370)

4.43.19 TCollection.Sort

Synopsis: Sort the items in the collection.

Declaration: `procedure Sort(const Compare: TCollectionSortCompare)`

Visibility: public

Description: `Sort` sorts the items in the collection, and uses the `Compare` procedure to compare 2 items in the collection. It is more efficient do use this method than to perform the sort manually, because the list items are manipulated directly.

For more information on how the `Compare` function should behave, see the TCollectionSortCompare (277) type.

See also: TCollectionSortCompare (277)

4.43.20 TCollection.Count

Synopsis: Number of items in the collection.

Declaration: `Property Count : Integer`

Visibility: public

Access: Read

Description: `Count` contains the number of items in the collection.

Remark The items in the collection are identified by their TCollectionItem.Index (375) property, which is a zero-based index, meaning that it can take values between `0` and `Count-1`, borders included.

See also: TCollectionItem.Index (375), TCollection.Items (371)

4.43.21 TCollection.ItemClass

Synopsis: Class pointer for each item in the collection.

Declaration: `Property ItemClass : TCollectionItemClass`

Visibility: public

Access: Read

Description: `ItemClass` is the class pointer with which each new item in the collection is created. It is the value that was passed to the collection's constructor when it was created, and does not change during the lifetime of the collection.

See also: TCollectionItem (373), TCollection.Items (371)

4.43.22 TCollection.Items

Synopsis: Indexed array of items in the collection.

Declaration: `Property Items[Index: Integer]: TCollectionItem`

Visibility: public

Access: Read,Write

Description: `Items` provides indexed access to the items in the collection. Since the array is zero-based, `Index` should be an integer between `0` and `Count-1`.

It is possible to set or retrieve an element in the array. When setting an element of the array, the object that is assigned should be compatible with the class of the objects in the collection, as given by the TCollection.ItemClass (371) property.

Adding an element to the array can be done with the TCollection.Add (367) method. The array can be cleared with the TCollection.Clear (368) method. Removing an element of the array should be done by freeing that element.

See also: TCollection.Count (371), TCollection.ItemClass (371), TCollection.Clear (368), TCollection.Add (367)

4.44 TCollectionEnumerator

4.44.1 Description

`TCollectionEnumerator` implements the #rtl.system.IEnumerator (1617) interface for the TCollection (365) class, so the `TCollection` class can be used in a `for ... in` loop. It is returned by the TCollection.GetEnumerator (369) method of `TCollection`.

See also: TCollection (365), TCollection.GetEnumerator (369), #rtl.system.IEnumerator (1617)

4.44.2 Method overview

Page	Method	Description
372	Create	Initialize a new instance of `TCollectionEnumerator`.
372	GetCurrent	Return the current pointer in the list.
372	MoveNext	Move the position of the enumerator to the next position in the collection.

4.44.3 Property overview

Page	Properties	Access	Description
373	Current	r	Current pointer in the list.

4.44.4 TCollectionEnumerator.Create

Synopsis: Initialize a new instance of `TCollectionEnumerator`.

Declaration:
```
constructor Create(ACollection: TCollection)
```

Visibility: public

Description: `Create` initializes a new instance of `TCollectionEnumerator` and keeps a reference to the collection `ACollection` that will be enumerated.

See also: TCollection (365)

4.44.5 TCollectionEnumerator.GetCurrent

Synopsis: Return the current pointer in the list.

Declaration:
```
function GetCurrent : TCollectionItem
```

Visibility: public

Description: `GetCurrent` returns the current TCollectionItem (373) instance in the enumerator.

Errors: No checking is done on the validity of the current position.

See also: MoveNext (372), TCollectionItem (373)

4.44.6 TCollectionEnumerator.MoveNext

Synopsis: Move the position of the enumerator to the next position in the collection.

Declaration:
```
function MoveNext : Boolean
```

Visibility: public

Description: `MoveNext` puts the pointer on the next item in the collection, and returns `True` if this succeeded, or `False` if the pointer is past the last element in the list.

Errors: Note that if `False` is returned, calling `GetCurrent` will result in an exception.

See also: GetCurrent (372)

4.44.7 TCollectionEnumerator.Current

Synopsis: Current pointer in the list.

Declaration: `Property Current : TCollectionItem`

Visibility: public

Access: Read

Description: `Current` redefines GetCurrent (372) as a property.

See also: GetCurrent (372)

4.45 TCollectionItem

4.45.1 Description

`TCollectionItem` and TCollection (365) form a pair of base classes that manage a collection of named objects. The `TCollectionItem` is the named object that is managed, it represents one item in the collection. An item in the collection is represented by three properties: TCollectionItem.DisplayName (375), TCollection.Index (365) and TCollectionItem.ID (375).

A `TCollectionItem` object is never created directly. To manage a set of named items, it is necessary to make a descendant of `TCollectionItem` to which needed properties and methods are added. This descendant can then be managed with a TCollection (365) class. The managing collection will create and destroy it's items by itself, it should therefore never be necessary to create `TCollectionItem` descendants manually.

See also: TCollection (365)

4.45.2 Method overview

Page	Method	Description
374	Create	Creates a new instance of this collection item.
374	Destroy	Destroys this collection item.
374	GetNamePath	Returns the namepath of this collection item.

4.45.3 Property overview

Page	Properties	Access	Description
374	Collection	rw	Pointer to the collection managing this item.
375	DisplayName	rw	Name of the item, displayed in the object inspector.
375	ID	r	Initial index of this item.
375	Index	rw	Index of the item in its managing collection TCollection.Items (371) property.

4.45.4 TCollectionItem.Create

Synopsis: Creates a new instance of this collection item.

Declaration: `constructor Create(ACollection: TCollection); Virtual`

Visibility: public

Description: `Create` instantiates a new item in a TCollection (365). It is called by the TCollection.Add (367) function and should under normal circumstances never be called directly. called

See also: TCollectionItem.Destroy (374)

4.45.5 TCollectionItem.Destroy

Synopsis: Destroys this collection item.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` removes the item from the managing collection and Destroys the item instance.
This is the only way to remove items from a collection;

See also: TCollectionItem.Create (374)

4.45.6 TCollectionItem.GetNamePath

Synopsis: Returns the namepath of this collection item.

Declaration: `function GetNamePath : string; Override`

Visibility: public

Description: `GetNamePath` overrides the TPersistent.GetNamePath (437) method to return the name of the managing collection and appends its TCollectionItem.Index (375) property.

See also: TCollectionItem.Collection (374), TPersistent.GetNamePath (437), TCollectionItem.Index (375)

4.45.7 TCollectionItem.Collection

Synopsis: Pointer to the collection managing this item.

Declaration: `Property Collection : TCollection`

Visibility: public

Access: Read,Write

Description: `Collection` points to the collection managing this item. This property can be set to point to a new collection. If this is done, the old collection will be notified that the item should no longer be managed, and the new collection is notified that it should manage this item as well.

See also: TCollection (365)

4.45.8 TCollectionItem.ID

Synopsis: Initial index of this item.

Declaration: `Property ID : Integer`

Visibility: public

Access: Read

Description: `ID` is the initial value of TCollectionItem.Index (375); it doesn't change after the index changes. It can be used to uniquely identify the item. The `ID` property doesn't change as items are added and removed from the collection.

While the TCollectionItem.Index (375) property forms a continuous series, `ID` does not. If items are removed from the collection, their `ID` is not used again, leaving gaps. Only when the collection is initially created, the `ID` and `Index` properties will be equal.

See also: TCollection.Items (371), TCollectionItem.Index (375)

4.45.9 TCollectionItem.Index

Synopsis: Index of the item in its managing collection TCollection.Items (371) property.

Declaration: `Property Index : Integer`

Visibility: public

Access: Read,Write

Description: `Index` is the current index of the item in its managing collection's TCollection.Items (371) property. This property may change as items are added and removed from the collection.

The index of an item is zero-based, i.e. the first item has index zero. The last item has index `Count-1` where `Count` is the number of items in the collection.

The `Index` property of the items in a collection form a continuous series ranging from `0` to `Count-1`. The TCollectionItem.ID (375) property does not form a continuous series, but can also be used to identify an item.

See also: TCollectionItem.ID (375), TCollection.Items (371)

4.45.10 TCollectionItem.DisplayName

Synopsis: Name of the item, displayed in the object inspector.

Declaration: `Property DisplayName : string`

Visibility: public

Access: Read,Write

Description: `DisplayName` contains the name of this item as shown in the object inspector. For `TCollectionItem` this returns always the class name of the managing collection, followed by the index of the item.

`TCollectionItem` does not implement any functionality to store the `DisplayName` property. The property can be set, but this will have no effect other than that the managing collection is notified of a change. The actual displayname will remain unchanged. To store the `DisplayName` property,`TCollectionItem` descendants should override the TCollectionItem.SetDisplayName (373) and TCollectionItem.GetDisplayName (373) to add storage functionality.

See also: TCollectionItem.Index (375), TCollectionItem.ID (375), TCollectionItem.GetDisplayName (373), TCollectionItem.SetDisplayName (373)

4.46 TComponent

4.46.1 Description

`TComponent` is the base class for any set of classes that needs owner-owned functionality, and which needs support for property streaming. All classes that should be handled by an IDE (Integrated Development Environment) must descend from `TComponent`, as it includes all support for streaming all its published properties.

Components can 'own' other components. `TComponent` introduces methods for enumerating the child components. It also allows to name the owned components with a unique name. Furthermore, functionality for sending notifications when a component is removed from the list or removed from memory altogether is also introduced in `TComponent`

`TComponent` introduces a form of automatic memory management: When a component is destroyed, all its child components will be destroyed first.

4.46.2 Interfaces overview

Page	Interfaces	Description
316	IInterfaceComponentReference	Interface for checking component references.
1619	IUnknown	Basic interface for all COM-based interfaces.

4.46.3 Method overview

Page	Method	Description
378	BeforeDestruction	Overrides standard BeforeDestruction.
378	Create	Creates a new instance of the component.
378	Destroy	Destroys the instance of the component.
378	DestroyComponents	Destroy child components.
379	Destroying	Called when the component is being destroyed.
379	ExecuteAction	Standard action execution method.
379	FindComponent	Finds and returns the named component in the owned components.
379	FreeNotification	Ask the component to notify called when it is being destroyed.
380	FreeOnRelease	Part of the `IVCLComObject` interface.
380	GetEnumerator	Create an `IEnumerator` instance.
380	GetNamePath	Returns the name path of this component.
380	GetParentComponent	Returns the parent component.
381	HasParent	Does the component have a parent ?
381	InsertComponent	Insert the given component in the list of owned components.
382	IsImplementorOf	Checks if the current component is the implementor of the interface.
377	Notification	Called by components that are freed and which received a FreeNotification.
382	ReferenceInterface	Interface implementation of Notification.
381	RemoveComponent	Remove the given component from the list of owned components.
380	RemoveFreeNotification	Remove a component from the Free Notification list.
381	SafeCallException	Part of the `IVCLComObject` Interface.
382	SetSubComponent	Sets the `csSubComponent` style.
382	UpdateAction	Updates the state of an action.
377	WriteState	Writes the component to a stream.

4.46.4 Property overview

Page	Properties	Access	Description
383	ComObject	r	Interface reference implemented by the component.
383	ComponentCount	r	Count of owned components.
383	ComponentIndex	rw	Index of component in it's owner's list.
383	Components	r	Indexed list (zero-based) of all owned components.
384	ComponentState	r	Current component's state.
384	ComponentStyle	r	Current component's style.
384	DesignInfo	rw	Information for IDE designer.
385	Name	rws	Name of the component.
385	Owner	r	Owner of this component.
385	Tag	rw	Tag value of the component.
385	VCLComObject	rw	Not implemented.

4.46.5 TComponent.Notification

Synopsis: Called by components that are freed and which received a FreeNotification.

Declaration:
```
procedure Notification(AComponent: TComponent; Operation: TOperation)
                       ;  Virtual
```

Visibility: protected

Description: `Notification` is called whenever a child component is destroyed, inserted or removed from the list of owned component. Components that were requested to send a notification when they are freed ((with FreeNotification (379)) will also call `Notification` when they are freed.

The `AComponent` parameter specifies which component sends the notification, and `Operation` specifies whether the component is being inserted into or removed from the child component list, or whether it is being destroyed.

Descendants of `TComponent` can use FreeNotification (379) to request notification of the destruction of another object. By overriding the `Notification` method, they can do special processing (typically, set a reference to this component to `Nil`) when this component is destroyed. The `Notification` method is called quite often in the streaming process, so speed should be a consideration when overriding this method.

See also: TOperation (284), TComponent.FreeNotification (379), TComponent.RemoveFreeNotification (380)

4.46.6 TComponent.WriteState

Synopsis: Writes the component to a stream.

Declaration:
```
procedure WriteState(Writer: TWriter);  Virtual
```

Visibility: public

Description: `WriteState` writes the component's current state to a stream through the writer (524) object `writer`. Values for all published properties of the component can be written to the stream. Normally there is no need to call `WriteState` directly. The streaming system calls `WriteState` itself.

The TComponent (376) implementation of `WriteState` simply calls TWriter.WriteData (524). Descendant classes can, however, override `WriteState`to provide additional processing of stream data.

See also: ReadState (376), TStream.WriteComponent (460), TWriter.WriteData (524)

4.46.7 TComponent.Create

Synopsis: Creates a new instance of the component.

Declaration: `constructor Create(AOwner: TComponent); Virtual`

Visibility: public

Description: `Create` creates a new instance of a `TComponent` class. If `AOwner` is not `Nil`, the new component attempts to insert itself in the list of owned components of the owner.

See also: Insert (376), Owner (385)

4.46.8 TComponent.Destroy

Synopsis: Destroys the instance of the component.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` sends a `opRemove` notification to all components in the free-notification list. After that, all owned components are destroyed by calling DestroyComponents (378) (and hence removed from the list of owned components). When this is done, the component removes itself from its owner's child component list. After that, the parent's destroy method is called.

See also: Notification (377), Owner (385), DestroyComponents (378), Components (383)

4.46.9 TComponent.BeforeDestruction

Synopsis: Overrides standard BeforeDestruction.

Declaration: `procedure BeforeDestruction; Override`

Visibility: public

Description: `BeforeDestruction` is overridden by `TComponent` to set the `csDestroying` flag in TComponent.ComponentState (384)

See also: TComponent.ComponentState (384)

4.46.10 TComponent.DestroyComponents

Synopsis: Destroy child components.

Declaration: `procedure DestroyComponents`

Visibility: public

Description: `DestroyComponents` calls the destructor of all owned components, till no more components are left in the Components (383) array.

Calling the destructor of an owned component has as the effect that the component will remove itself from the list of owned components, if nothing has disrupted the sequence of destructors.

Errors: If an overridden 'destroy' method does not call it's inherited destructor or raises an exception, it's TComponent.Destroy (378) destructor will not be called, which may result in an endless loop.

See also: Destroy (378), Components (383)

4.46.11 TComponent.Destroying

Synopsis: Called when the component is being destroyed.

Declaration: `procedure Destroying`

Visibility: public

Description: `Destroying` sets the `csDestroying` flag in the component's state (376) property, and does the same for all owned components.

It is not necessary to call `Destroying` directly, the destructor Destroy (378) does this automatically.

See also: State (376), Destroy (378)

4.46.12 TComponent.ExecuteAction

Synopsis: Standard action execution method.

Declaration: `function ExecuteAction(Action: TBasicAction) : Boolean;  Dynamic`

Visibility: public

Description: `ExecuteAction` checks whether `Action` handles the current component, and if yes, calls the `ExecuteAction` method, passing itself as a parameter. The function returns `True` if the action handles the current component.

See also: TBasicAction (339), TBasicAction.ExecuteAction (339), TBasicAction.HandlesTarget (340), TComponent.UpdateAction (382)

4.46.13 TComponent.FindComponent

Synopsis: Finds and returns the named component in the owned components.

Declaration: `function FindComponent(const AName: string) : TComponent`

Visibility: public

Description: `FindComponent` searches the component with name `AName` in the list of owned components. If `AName` is empty, then `Nil` is returned.

See also: Components (383), Name (385)

4.46.14 TComponent.FreeNotification

Synopsis: Ask the component to notify called when it is being destroyed.

Declaration: `procedure FreeNotification(AComponent: TComponent)`

Visibility: public

Description: `FreeNotification` inserts `AComponent` in the freenotification list. When the component is destroyed, the Notification (377) method is called for all components in the freenotification list.

See also: Components (383), Notification (377), TComponent.RemoveFreeNotification (380)

4.46.15 TComponent.RemoveFreeNotification

Synopsis: Remove a component from the Free Notification list.

Declaration: `procedure RemoveFreeNotification(AComponent: TComponent)`

Visibility: public

Description: `RemoveFreeNotification` removes `AComponent` from the freenotification list.

See also: TComponent.FreeNotification (379)

4.46.16 TComponent.FreeOnRelease

Synopsis: Part of the `IVCLComObject` interface.

Declaration: `procedure FreeOnRelease`

Visibility: public

Description: Provided for Delphi compatibility, but is not yet implemented.

4.46.17 TComponent.GetEnumerator

Synopsis: Create an `IEnumerator` instance.

Declaration: `function GetEnumerator : TComponentEnumerator`

Visibility: public

Description: `GetEnumerator` is the implementation of the IEnumerable (1616) interface for `TComponent`. It creates a TComponentEnumerator (386) instance and returns it's IEnumerator (1617) interface. The enumerator enumerates all child components of the component instance.

See also: TComponentEnumerator (386), IEnumerator (1617), IEnumerable (1616)

4.46.18 TComponent.GetNamePath

Synopsis: Returns the name path of this component.

Declaration: `function GetNamePath : string;  Override`

Visibility: public

Description: `GetNamePath` returns the name of the component as it will be shown in the object inspector.

`TComponent` overrides `GetNamePath` so it returns the Name (385) property of the component.

See also: Name (385), TPersistent.GetNamePath (437)

4.46.19 TComponent.GetParentComponent

Synopsis: Returns the parent component.

Declaration: `function GetParentComponent : TComponent;  Dynamic`

Visibility: public

Description: `GetParentComponent` can be implemented to return the parent component of this component. The implementation of this method in `TComponent` always returns `Nil`. Descendant classes must override this method to return the visual parent of the component.

See also: HasParent (381), Owner (385)

4.46.20 TComponent.HasParent

Synopsis: Does the component have a parent ?

Declaration: `function HasParent : Boolean;  Dynamic`

Visibility: public

Description: `HasParent` can be implemented to return whether the parent of the component exists. The implementation of this method in TComponent always returns `False`, and should be overridden by descendant classes to return `True` when a parent is available. If `HasParent` returns `True`, then GetParentComponent (380) will return the parent component.

See also: HasParent (381), Owner (385)

4.46.21 TComponent.InsertComponent

Synopsis: Insert the given component in the list of owned components.

Declaration: `procedure InsertComponent(AComponent: TComponent)`

Visibility: public

Description: `InsertComponent` attempts to insert `AComponent` in the list with owned components. It first calls ValidateComponent (376) to see whether the component can be inserted. It then checks whether there are no name conflicts by calling ValidateRename (376). If neither of these checks have raised an exception the component is inserted, and notified of the insert.

See also: RemoveComponent (381), Insert (376), ValidateContainer (376), ValidateRename (376), Notification (377)

4.46.22 TComponent.RemoveComponent

Synopsis: Remove the given component from the list of owned components.

Declaration: `procedure RemoveComponent(AComponent: TComponent)`

Visibility: public

Description: `RemoveComponent` will send an `opRemove` notification to `AComponent` and will then proceed to remove `AComponent` from the list of owned components.

See also: InsertComponent (381), Remove (376), ValidateRename (376), Notification (377)

4.46.23 TComponent.SafeCallException

Synopsis: Part of the `IVCLComObject` Interface.

Declaration:
```
function SafeCallException(ExceptObject: TObject;
                           ExceptAddr: CodePointer) : HRESULT;  Override
```

Visibility: public

Description: Provided for Delphi compatibility, but not implemented.

4.46.24 TComponent.SetSubComponent

Synopsis: Sets the `csSubComponent` style.

Declaration: `procedure SetSubComponent(ASubComponent: Boolean)`

Visibility: public

Description: `SetSubComponent` includes `csSubComponent` in the ComponentStyle (384) property if `ASubComponent` is `True`, and excludes it again if `ASubComponent` is `False`.

See also: TComponent.ComponentStyle (384)

4.46.25 TComponent.UpdateAction

Synopsis: Updates the state of an action.

Declaration: `function UpdateAction(Action: TBasicAction) : Boolean; Dynamic`

Visibility: public

Description: `UpdateAction` checks whether `Action` handles the current component, and if yes, calls the `UpdateTarget` method, passing itself as a parameter. The function returns `True` if the action handles the current component.

See also: TBasicAction (339), TBasicAction.UpdateTarget (340), TBasicAction.HandlesTarget (340), TBasicAction.ExecuteAction (339)

4.46.26 TComponent.IsImplementorOf

Synopsis: Checks if the current component is the implementor of the interface.

Declaration: `function IsImplementorOf(const Intf: IInterface) : Boolean`

Visibility: public

Description: `IsImplementorOf` returns `True` if the current component implements the given interface. The interface should descend from IInterfaceComponentReference (316) and the `GetComponent` method should return the current instance.

See also: IInterfaceComponentReference (316)

4.46.27 TComponent.ReferenceInterface

Synopsis: Interface implementation of Notification.

Declaration: `procedure ReferenceInterface(const intf: IInterface; op: TOperation)`

Visibility: public

Description: `ReferenceInterface` can be used to notify an interface of a component operation: it is the equivalent of the TComponent.Notification (377) method of `TComponent` for interfaces. If the interface implements IInterfaceComponentReference (316), then the component that implements the interface is notified of the given operation `Op`.

Errors: None.

See also: TComponent.Notification (377), IInterfaceComponentReference (316)

4.46.28 TComponent.ComObject

Synopsis: Interface reference implemented by the component.

Declaration: `Property ComObject : IUnknown`

Visibility: public

Access: Read

Description: `ComObject` returns the COM interface represented by the component. If the component does not represent a COM interface, reading this property will raise an EComponentError (310).

See also: EComponentError (310)

4.46.29 TComponent.Components

Synopsis: Indexed list (zero-based) of all owned components.

Declaration: `Property Components[Index: Integer]: TComponent`

Visibility: public

Access: Read

Description: `Components` provides indexed access to the list of owned components. `Index` can range from 0 to ComponentCount-1 (383).

See also: ComponentCount (383), Owner (385)

4.46.30 TComponent.ComponentCount

Synopsis: Count of owned components.

Declaration: `Property ComponentCount : Integer`

Visibility: public

Access: Read

Description: `ComponentCount` returns the number of components that the current component owns. It can be used to determine the valid index range in the Component (383) array.

See also: Components (383), Owner (385)

4.46.31 TComponent.ComponentIndex

Synopsis: Index of component in it's owner's list.

Declaration: `Property ComponentIndex : Integer`

Visibility: public

Access: Read,Write

Description: `ComponentIndex` is the index of the current component in its owner's list of components. If the component has no owner, the value of this property is -1.

See also: Components (383), ComponentCount (383), Owner (385)

4.46.32 TComponent.ComponentState

Synopsis: Current component's state.

Declaration: `Property ComponentState : TComponentState`

Visibility: public

Access: Read

Description: `ComponentState` indicates the current state of the component. It is a set of flags which indicate the various stages in the lifetime of a component. The following values can occur in this set:

Table 4.25: Component states

Flag	Meaning
csLoading	The component is being loaded from stream
csReading	Component properties are being read from stream.
csWriting	Component properties are being written to stream.
csDestroying	The component or one of it's owners is being destroyed.
csAncestor	The component is being streamed as part of a frame
csUpdating	The component is being updated
csFixups	References to other components are being resolved
csFreeNotification	The component has freenotifications.
csInline	The component is being loaded as part of a frame
csDesignInstance	? not used.

The component state is set by various actions such as reading it from stream, destroying it etc.

See also: SetAncestor (376), SetDesigning (376), SetInline (376), SetDesignInstance (376), Updating (376), Updated (376), Loaded (376)

4.46.33 TComponent.ComponentStyle

Synopsis: Current component's style.

Declaration: `Property ComponentStyle : TComponentStyle`

Visibility: public

Access: Read

Description: Current component's style.

4.46.34 TComponent.DesignInfo

Synopsis: Information for IDE designer.

Declaration: `Property DesignInfo : LongInt`

Visibility: public

Access: Read,Write

Description: `DesignInformation` can be used by an IDE to store design information in the component. It should not be used by an application programmer.

See also: Tag (385)

4.46.35 TComponent.Owner

Synopsis: Owner of this component.

Declaration: `Property Owner : TComponent`

Visibility: public

Access: Read

Description: `Owner` returns the owner of this component. The owner cannot be set except by explicitly inserting the component in another component's owned components list using that component's InsertComponent (381) method, or by removing the component from it's owner's owned component list using the RemoveComponent (381) method.

See also: Components (383), InsertComponent (381), RemoveComponent (381)

4.46.36 TComponent.VCLComObject

Synopsis: Not implemented.

Declaration: `Property VCLComObject : Pointer`

Visibility: public

Access: Read,Write

Description: `VCLComObject` is not yet implemented in Free Pascal.

4.46.37 TComponent.Name

Synopsis: Name of the component.

Declaration: `Property Name : TComponentName`

Visibility: published

Access: Read,Write

Description: `Name` is the name of the component. This name should be a valid identifier, i.e. must start with a letter or underscore, and can contain only letters, numbers and the underscore character. When attempting to set the name of a component, the name will be checked for validity. Furthermore, when a component is owned by another component, the name must be either empty or must be unique among the child component names.

By "letters", 7-bit letters are meant.

Errors: Attempting to set the name to an invalid value will result in an exception being raised.

See also: ValidateRename (376), Owner (385)

4.46.38 TComponent.Tag

Synopsis: Tag value of the component.

Declaration: `Property Tag : PtrInt`

Visibility: published

Access: Read,Write

Description: `Tag` can be used to store an integer value in the component. This value is streamed together with all other published properties. It can be used for instance to quickly identify a component in an event handler.

See also: Name (385)

4.47 TComponentEnumerator

4.47.1 Description

`TComponentEnumerator` implements the #rtl.system.IEnumerator (1617) interface for the TComponent (376) class, so the `TComponent` class can be used in a `for ... in` loop over the TComponent.Components (383) child components of the component. It is returned by the TComponent.GetEnumerator (380) method of `TComponent`.

See also: TComponent (376), TComponent.GetEnumerator (380), #rtl.system.IEnumerator (1617)

4.47.2 Method overview

Page	Method	Description
386	Create	Initialize a new instance of `TComponentEnumerator`.
386	GetCurrent	Return the current pointer in the list.
387	MoveNext	Move the position of the enumerator to the next position in the children of the component.

4.47.3 Property overview

Page	Properties	Access	Description
387	Current	r	Current pointer in the list.

4.47.4 TComponentEnumerator.Create

Synopsis: Initialize a new instance of `TComponentEnumerator`.

Declaration: `constructor Create(AComponent: TComponent)`

Visibility: public

Description: `Create` initializes a new instance of `TComponentEnumerator` and keeps a reference to the component `AComponent` that will be enumerated.

See also: TComponent (376)

4.47.5 TComponentEnumerator.GetCurrent

Synopsis: Return the current pointer in the list.

Declaration: `function GetCurrent : TComponent`

Visibility: public

Description: `GetCurrent` returns the current TComponent (376) child component instance in the enumerator.

Errors: No checking is done on the validity of the current position.

See also: MoveNext (387), TComponent.Components (383)

4.47.6 TComponentEnumerator.MoveNext

Synopsis: Move the position of the enumerator to the next position in the children of the component.

Declaration: `function MoveNext : Boolean`

Visibility: public

Description: `MoveNext` puts the pointer on the next child in the components child components, and returns `True` if this succeeded, or `False` if the pointer is past the last child in the list.

Errors: Note that if `False` is returned, calling `GetCurrent` will result in an exception.

See also: GetCurrent (386)

4.47.7 TComponentEnumerator.Current

Synopsis: Current pointer in the list.

Declaration: `Property Current : TComponent`

Visibility: public

Access: Read

Description: `Current` redefines GetCurrent (386) as a property.

See also: GetCurrent (386)

4.48 TCustomMemoryStream

4.48.1 Description

`TCustomMemoryStream` is the parent class for streams that stored their data in memory. It introduces all needed functions to handle reading from and navigating through the memory, and introduces a Memory (389) property which points to the memory area where the stream data is kept.

The only thing which `TCustomMemoryStream` does not do is obtain memory to store data when writing data or the writing of data. This functionality is implemented in descendant streams such as TMemoryStream (425). The reason for this approach is that this way it is possible to create e.g. read-only descendants of `TCustomMemoryStream` that point to a fixed part in memory which can be read from, but not written to.

Remark Since `TCustomMemoryStream` is an abstract class, do not create instances of `TMemoryStream` directly. Instead, create instances of descendants such as TMemoryStream (425).

See also: TMemoryStream (425), TStream (455)

4.48.2 Method overview

Page	Method	Description
388	Read	Reads `Count` bytes from the stream into `buffer`.
389	SaveToFile	Writes the contents of the stream to a file.
388	SaveToStream	Writes the contents of the memory stream to another stream.
388	Seek	Sets a new position in the stream.

4.48.3 Property overview

Page	Properties	Access	Description
389	Memory	r	Pointer to the data kept in the memory stream.

4.48.4 TCustomMemoryStream.Read

Synopsis: Reads `Count` bytes from the stream into `buffer`.

Declaration:
```
function Read(var Buffer; Count: LongInt) : LongInt;  Override
```

Visibility: public

Description: `Read` reads `Count` bytes from the stream into the memory pointed to by `buffer`. It returns the number of bytes actually read.

This method overrides the TStream.Read (456) method of TStream (455). It will read as much bytes as are still available in the memory area pointer to by Memory (389). After the bytes are read, the internal stream position is updated.

See also: TCustomMemoryStream.Memory (389), TStream.Read (456)

4.48.5 TCustomMemoryStream.Seek

Synopsis: Sets a new position in the stream.

Declaration:
```
function Seek(const Offset: Int64; Origin: TSeekOrigin) : Int64
                  ;  Override
```

Visibility: public

Description: `Seek` overrides the abstract TStream.Seek (457) method. It simply updates the internal stream position, and returns the new position.

Errors: No checking is done whether the new position is still a valid position, i.e. whether the position is still within the range `0..Size`. Attempting a seek outside the valid memory range of the stream may result in an exception at the next read or write operation.

See also: TStream.Position (464), TStream.Size (465), TCustomMemoryStream.Memory (389)

4.48.6 TCustomMemoryStream.SaveToStream

Synopsis: Writes the contents of the memory stream to another stream.

Declaration:
```
procedure SaveToStream(Stream: TStream)
```

Visibility: public

Description: `SaveToStream` writes the contents of the memory stream to `Stream`. The content of `Stream` is not cleared first. The current position of the memory stream is not changed by this action.

Remark This method will work much faster than the use of the TStream.CopyFrom (459) method:

```
Seek(0,soFromBeginning);
  Stream.CopyFrom(Self,Size);
```

because the `CopyFrom` method copies the contents in blocks, while `SaveToStream` writes the contents of the memory as one big block.

Errors: If an error occurs when writing to `Stream` an EStreamError (312) exception will be raised.

See also: TCustomMemoryStream.SaveToFile (389), TStream.CopyFrom (459)

4.48.7 TCustomMemoryStream.SaveToFile

Synopsis: Writes the contents of the stream to a file.

Declaration: `procedure SaveToFile(const FileName: string)`

Visibility: public

Description: `SaveToFile` writes the contents of the stream to a file with name `FileName`. It simply creates a filestream and writes the contents of the memorystream to this file stream using TCustomMemoryStream.SaveToStream (388).

Remark This method will work much faster than the use of the TStream.CopyFrom (459) method:

```
Stream:=TFileStream.Create(fmCreate,FileName);
  Seek(0,soFromBeginning);
  Stream.CopyFrom(Self,Size);
```

because the `CopyFrom` method copies the contents in blocks, while `SaveToFile` writes the contents of the memory as one big block.

Errors: If an error occurs when creating or writing to the file, an EStreamError (312) exception may occur.

See also: TCustomMemoryStream.SaveToStream (388), TFileStream (395), TStream.CopyFrom (459)

4.48.8 TCustomMemoryStream.Memory

Synopsis: Pointer to the data kept in the memory stream.

Declaration: `Property Memory : Pointer`

Visibility: public

Access: Read

Description: `Memory` points to the memory area where stream keeps it's data. The property is read-only, so the pointer cannot be set this way.

Remark Do not write to the memory pointed to by `Memory`, since the memory content may be read-only, and thus writing to it may cause errors.

See also: TStream.Size (465)

4.49 TDataModule

4.49.1 Description

`TDataModule` is a container for non-visual objects which can be used in an IDE to group non-visual objects which can be used by various other containers (forms) in a project. Notably, data access components are typically stored on a datamodule. Web components and services can also be implemented as descendants of datamodules.

`TDataModule` introduces some events which make it easier to program, and provides the needed streaming capabilities for persistent storage.

An IDE will typically allow to create a descendant of `TDataModule` which contains non-visual components in it's published property list.

See also: TDataModule.OnCreate (392)

4.49.2 Method overview

Page	Method	Description
391	AfterConstruction	Overrides standard TObject (1623) behaviour.
391	BeforeDestruction	
390	Create	Create a new instance of a `TDataModule`.
390	CreateNew	
391	Destroy	Destroys the `TDataModule` instance.

4.49.3 Property overview

Page	Properties	Access	Description
391	DesignOffset	rw	Position property needed for manipulation in an IDE.
392	DesignPPI	rw	Design Time PPI.
392	DesignSize	rw	Size property needed for manipulation in an IDE.
393	OldCreateOrder	rw	Determines when `OnCreate` and `OnDestroy` are triggered.
392	OnCreate	rw	Event handler, called when the datamodule is created.
392	OnDestroy	rw	Event handler, called when the datamodule is destroyed.

4.49.4 TDataModule.Create

Synopsis: Create a new instance of a `TDataModule`.

Declaration:
```
constructor Create(AOwner: TComponent);  Override
```

Visibility: public

Description: `Create` creates a new instance of the `TDatamodule` and calls TDatamodule.CreateNew (390). After that it reads the published properties from a stream using InitInheritedComponent (300) if a descendant class is instantiated. If the OldCreateOrder (393) property is `True`, the TDataModule.OnCreate (392) event is called.

Errors: An exception can be raised during the streaming operation.

See also: TDataModule.CreateNew (390)

4.49.5 TDataModule.CreateNew

Synopsis:

Declaration:
```
constructor CreateNew(AOwner: TComponent)
constructor CreateNew(AOwner: TComponent; CreateMode: Integer);  Virtual
```

Visibility: public

Description: `CreateNew` creates a new instance of the class, but bypasses the streaming mechanism. The `CreateMode` parameter (by default zero) is not used in `TDataModule`. If the AddDataModule (293) handler is set, then it is called, with the newly created instance as an argument.

See also: TDataModule.Create (390), AddDataModule (293), TDataModule.OnCreate (392)

4.49.6 TDataModule.Destroy

Synopsis: Destroys the `TDataModule` instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` destroys the `TDataModule` instance. If the OldCreateOrder (393) property is `True` the OnDestroy (392) event handler is called prior to destroying the data module.

Before calling the inherited destroy, the RemoveDataModule (294) handler is called if it is set, and `Self` is passed as a parameter.

Errors: An event can be raised during the `OnDestroy` event handler.

See also: TDataModule.OnDestroy (392), RemoveDataModule (294)

4.49.7 TDataModule.AfterConstruction

Synopsis: Overrides standard TObject (1623) behaviour.

Declaration: `procedure AfterConstruction;  Override`

Visibility: public

Description: `AfterConstruction` calls the OnCreate (392) handler if the OldCreateOrder (393) property is `False`.

See also: TDataModule.OldCreateOrder (393), TDataModule.OnCreate (392)

4.49.8 TDataModule.BeforeDestruction

Synopsis:

Declaration: `procedure BeforeDestruction;  Override`

Visibility: public

Description: `BeforeDestruction` calls the OnDestroy (392) handler if the OldCreateOrder (393) property is `False`.

See also: TDataModule.OldCreateOrder (393), TDataModule.OnDestroy (392)

4.49.9 TDataModule.DesignOffset

Synopsis: Position property needed for manipulation in an IDE.

Declaration: `Property DesignOffset : TPoint`

Visibility: public

Access: Read,Write

Description: `DesignOffset` is the position of the datamodule when displayed in an IDE. It is streamed to the form file, and should not be used at run-time.

See also: TDataModule.DesignSize (392)

4.49.10 TDataModule.DesignSize

Synopsis: Size property needed for manipulation in an IDE.

Declaration: `Property DesignSize : TPoint`

Visibility: public

Access: Read,Write

Description: `DesignSize` is the size of the datamodule when displayed in an IDE. It is streamed to the form file, and should not be used at run-time.

See also: TDataModule.DesignOffset (391)

4.49.11 TDataModule.DesignPPI

Synopsis: Design Time PPI.

Declaration: `Property DesignPPI : Integer`

Visibility: public

Access: Read,Write

Description: `DesignPPI` exists for the benefit of graphical designers, to store the PPI at which the module was designed. It is not used in FPC itself.

4.49.12 TDataModule.OnCreate

Synopsis: Event handler, called when the datamodule is created.

Declaration: `Property OnCreate : TNotifyEvent`

Visibility: published

Access: Read,Write

Description: The `OnCreate` event is triggered when the datamodule is created and streamed. The exact moment of triggering is dependent on the value of the OldCreateOrder (393) property.

See also: TDataModule.Create (390), TDataModule.CreateNew (390), TDataModule.OldCreateOrder (393)

4.49.13 TDataModule.OnDestroy

Synopsis: Event handler, called when the datamodule is destroyed.

Declaration: `Property OnDestroy : TNotifyEvent`

Visibility: published

Access: Read,Write

Description: The `OnDestroy` event is triggered when the datamodule is destroyed The exact moment of triggering is dependent on the value of the OldCreateOrder (393) property.

See also: TDataModule.Destroy (391), TDataModule.OnCreate (392), TDataModule.Create (390), TDataModule.CreateNew (390), TDataModule.OldCreateOrder (393)

4.49.14 TDataModule.OldCreateOrder

Synopsis: Determines when `OnCreate` and `OnDestroy` are triggered.

Declaration: `Property OldCreateOrder : Boolean`

Visibility: published

Access: Read,Write

Description: `OldCreateOrder` determines when exactly the OnCreate (392) and OnDestroy (392) event handlers are called.

If set to `True`, then the `OnCreate` event handler is called after the data module was streamed. If it is set to `False`, then the handler is called prior to the streaming process.

If set to `True`, then the `OnDestroy` event handler is called before the data module is removed from the streaming system. If it is set to `False`, then the handler is called after the data module was removed from the streaming process.

See also: TDataModule.OnDestroy (392), TDataModule.OnCreate (392), TDataModule.Destroy (391), TDataModule.Create (390), TDataModule.CreateNew (390), TDataModule.OldCreateOrder (393)

4.50 TFiler

4.50.1 Description

Class responsible for streaming of components.

4.50.2 Method overview

Page	Method	Description
394	DefineBinaryProperty	
393	DefineProperty	
394	FlushBuffer	Flush the buffer.

4.50.3 Property overview

Page	Properties	Access	Description
395	Ancestor	rw	Ancestor component from which an inherited component is streamed.
395	IgnoreChildren	rw	Determines whether children will be streamed as well.
394	LookupRoot	r	Component used to look up ancestor components.
394	Root	rw	The root component is the initial component which is being streamed.

4.50.4 TFiler.DefineProperty

Synopsis:

Declaration:
```
procedure DefineProperty(const Name: string; ReadData: TReaderProc;
                         WriteData: TWriterProc; HasData: Boolean)
                         ;  Virtual;  Abstract
```

Visibility: public

Description:

4.50.5 TFiler.DefineBinaryProperty

Synopsis:

Declaration:
```
procedure DefineBinaryProperty(const Name: string;
                               ReadData: TStreamProc;
                               WriteData: TStreamProc; HasData: Boolean)
                               ;  Virtual;  Abstract
```

Visibility: public

Description:

4.50.6 TFiler.FlushBuffer

Synopsis: Flush the buffer.

Declaration:
```
procedure FlushBuffer;  Virtual;  Abstract
```

Visibility: public

Description: `FlushBuffer` flushes the buffer by calling the flushbuffer on the driver. It is provided for Delphi compatibility, and is not used in FPC.

See also: TAbstractObjectReader.FlushBuffer (326)

4.50.7 TFiler.Root

Synopsis: The root component is the initial component which is being streamed.

Declaration:
```
Property Root : TComponent
```

Visibility: public

Access: Read,Write

Description: The streaming process will stream a component and all the components which it owns. The `Root` component is the component which is initially streamed.

See also: LookupRoot (394)

4.50.8 TFiler.LookupRoot

Synopsis: Component used to look up ancestor components.

Declaration:
```
Property LookupRoot : TComponent
```

Visibility: public

Access: Read

Description: When comparing inherited component's values against parent values, the values are compared with the component in `LookupRoot`. Initially, it is set to Root (394).

See also: Root (394)

4.50.9 TFiler.Ancestor

Synopsis: Ancestor component from which an inherited component is streamed.

Declaration: `Property Ancestor : TPersistent`

Visibility: public

Access: Read,Write

Description: When streaming a component, this is the parent component. Only properties that differ from the parent's property value will be streamed.

See also: Root (394), LookupRoot (394)

4.50.10 TFiler.IgnoreChildren

Synopsis: Determines whether children will be streamed as well.

Declaration: `Property IgnoreChildren : Boolean`

Visibility: public

Access: Read,Write

Description: By default, all children (i.e. owned objects) will also be streamed when streaming a component. This property can be used to prevent owned objects from being streamed.

4.51 TFileStream

4.51.1 Description

`TFileStream` is a TStream (455) descendant that stores or reads it's data from a named file in the file system of the operating system.

To this end, it overrides some of the methods in `TStream` and implements them for the case of files on disk, and it adds the FileName (397) property to the list of public properties.

See also: TFileStream.Create (396), TStream (455)

4.51.2 Method overview

Page	Method	Description
396	Create	Creates a file stream.
396	Destroy	Destroys the file stream.

4.51.3 Property overview

Page	Properties	Access	Description
397	FileName	r	The filename of the stream.

4.51.4 TFileStream.Create

Synopsis: Creates a file stream.

Declaration:
```
constructor Create(const AFileName: string; Mode: Word)
constructor Create(const AFileName: string; Mode: Word;
                   Rights: Cardinal)
```

Visibility: public

Description: `Create` creates a new instance of a `TFileStream` class. It opens the file `AFileName` with mode `Mode`, which can have one of the following values:

Table 4.26:

fmCreate	TFileStream.Create (396) creates a new file if needed.
fmOpenRead	TFileStream.Create (396) opens a file with read-only access.
fmOpenWrite	TFileStream.Create (396) opens a file with write-only access.
fmOpenReadWrite	TFileStream.Create (396) opens a file with read-write access.

These constants (except `fmCreate`) can be OR-ed with the following to specify how sharing and file locking is supposed to be handled:

Table 4.27:

fmShareCompat	Open file in DOS share-compatibility mode.
fmShareExclusive	Lock file for exclusive use.
fmShareDenyWrite	Lock file so other processes can only read.
fmShareDenyRead	Lock file so other processes cannot read.
fmShareDenyNone	Do not lock file.

Note that sharing is advisory on Unix-like platforms.

After the file has been opened in the requested mode and a handle has been obtained from the operating system, the inherited constructor is called.

Errors: If the file could not be opened in the requested mode, an EFOpenError (310) exception is raised.

See also: TStream (455), TFileStream.FileName (397), THandleStream.Create (406)

4.51.5 TFileStream.Destroy

Synopsis: Destroys the file stream.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` closes the file (causing possible buffered data to be written to disk) and then calls the inherited destructor.

Do not call `destroy` directly, instead call the `Free` method. Destroy does not check whether `Self` is nil, while `Free` does.

See also: TFileStream.Create (396)

4.51.6 TFileStream.FileName

Synopsis: The filename of the stream.

Declaration: `Property FileName : string`

Visibility: public

Access: Read

Description: `FileName` is the name of the file that the stream reads from or writes to. It is the name as passed in the constructor of the stream; it cannot be changed. To write to another file, the stream must be freed and created again with the new filename.

See also: TFileStream.Create (396)

4.52 TFPList

4.52.1 Description

`TFPList` is a class that can be used to manage collections of pointers. It introduces methods and properties to store the pointers, search in the list of pointers, sort them. It manages its memory by itself, no intervention for that is needed. Contrary to TList (415), `TFPList` has no notification mechanism. If no notification mechanism is used, it is better to use `TFPList` instead of `TList`, as the performance of `TFPList` is much higher.

To manage collections of strings, it is better to use a TStrings (475) descendant such as TStringList (470). To manage general objects, a TCollection (365) class exists, from which a descendant can be made to manage collections of various kinds.

See also: TStrings (475), TCollection (365)

4.52.2 Method overview

Page	Method	Description
398	Add	Adds a new pointer to the list.
398	AddList	Add all pointers from another list.
402	Assign	Assign performs the given operation on the list.
398	Clear	Clears the pointer list.
399	Delete	Removes a pointer from the list.
398	Destroy	Destroys the list and releases the memory used to store the list elements.
399	Error	Raises an EListError (275) exception.
399	Exchange	Exchanges two pointers in the list.
399	Expand	Increases the capacity of the list if needed.
400	Extract	Remove the first occurrence of a pointer from the list.
400	First	Returns the first non-nil pointer in the list.
403	ForEachCall	Call a procedure or method for each pointer in the list.
400	GetEnumerator	Create an `IEnumerator` instance.
400	IndexOf	Returns the index of a given pointer.
401	IndexOfItem	Search an item in the list.
401	Insert	Inserts a new pointer in the list at a given position.
401	Last	Returns the last non-nil pointer in the list.
401	Move	Moves a pointer from one position in the list to another.
402	Pack	Removes `Nil` pointers from the list and frees unused memory.
402	Remove	Removes a value from the list.
402	Sort	Sorts the pointers in the list.

4.52.3 Property overview

Page	Properties	Access	Description
403	Capacity	rw	Current capacity (i.e. number of pointers that can be stored) of the list.
403	Count	rw	Current number of pointers in the list.
404	Items	rw	Provides access to the pointers in the list.
404	List	r	Memory array where pointers are stored.

4.52.4 TFPList.Destroy

Synopsis: Destroys the list and releases the memory used to store the list elements.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` destroys the list and releases the memory used to store the list elements. The elements themselves are in no way touched, i.e. any memory they point to must be explicitly released before calling the destructor.

4.52.5 TFPList.AddList

Synopsis: Add all pointers from another list.

Declaration: `procedure AddList(AList: TFPList)`

Visibility: public

Description: `AddList` adds all pointers from `AList` to the list. If a pointer is already present, it is added a second time.

See also: TFPList.Assign (402), TList.AddList (418)

4.52.6 TFPList.Add

Synopsis: Adds a new pointer to the list.

Declaration: `function Add(Item: Pointer) : Integer`

Visibility: public

Description: `Add` adds a new pointer to the list after the last pointer (i.e. at position `Count`, thus increasing the item count with 1. If the list is at full capacity, the capacity of the list is expanded, using the Expand (399) method.

To insert a pointer at a certain position in the list, use the Insert (401) method instead.

See also: Delete (399), Grow (415), Insert (401)

4.52.7 TFPList.Clear

Synopsis: Clears the pointer list.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` removes all pointers from the list, and sets the capacity to 0, thus freeing any memory allocated to maintain the list.

See also: Destroy (398)

4.52.8 TFPList.Delete

Synopsis: Removes a pointer from the list.

Declaration: `procedure Delete(Index: Integer)`

Visibility: public

Description: `Delete` removes the pointer at position `Index` from the list, shifting all following pointers one position up (or to the left).

The memory the pointer is pointing to is *not* deallocated.

4.52.9 TFPList.Error

Synopsis: Raises an EListError (275) exception.

Declaration: `class procedure Error(const Msg: string; Data: PtrInt)`

Visibility: public

Description: `Error` raises an EListError (275) exception, with a message formatted with `Msg` and `Data`.

4.52.10 TFPList.Exchange

Synopsis: Exchanges two pointers in the list.

Declaration: `procedure Exchange(Index1: Integer; Index2: Integer)`

Visibility: public

Description: `Exchange` exchanges the pointers at positions `Index1` and `Index2`. Both pointers must be withing the current range of the list, or an EListError (275) exception will be raised.

4.52.11 TFPList.Expand

Synopsis: Increases the capacity of the list if needed.

Declaration: `function Expand : TFPList`

Visibility: public

Description: `Expand` increases the capacity of the list if the current element count matches the current list capacity.

The capacity is increased according to the following algorithm:

1. If the capacity is less than 3, the capacity is increased with 4.
2. If the capacity is larger than 3 and less than 8, the capacity is increased with 8.
3. If the capacity is larger than 8, the capacity is increased with 16.

The return value is `Self`.

See also: Capacity (403)

4.52.12 TFPList.Extract

Synopsis: Remove the first occurrence of a pointer from the list.

Declaration: `function Extract(Item: Pointer) : Pointer`

Visibility: public

Description: `Extract` searches for the first occurrence of `Item` in the list and deletes it from the list. If `Item` was found, it's value is returned. If `Item` was not found, `Nil` is returned.

See also: TFPList.Delete (399)

4.52.13 TFPList.First

Synopsis: Returns the first non-nil pointer in the list.

Declaration: `function First : Pointer`

Visibility: public

Description: `First` returns the value of the first non-nil pointer in the list.

If there are no pointers in the list or all pointers equal `Nil`, then `Nil` is returned.

See also: Last (401)

4.52.14 TFPList.GetEnumerator

Synopsis: Create an `IEnumerator` instance.

Declaration: `function GetEnumerator : TFPListEnumerator`

Visibility: public

Description: `GetEnumerator` is the implementation of the IEnumerable (1616) interface for `TFPList`. It creates a TFPListEnumerator (404) instance and returns it's IEnumerator (1617) interface.

See also: TFPListEnumerator (404), IEnumerator (1617), IEnumerable (1616)

4.52.15 TFPList.IndexOf

Synopsis: Returns the index of a given pointer.

Declaration: `function IndexOf(Item: Pointer) : Integer`

Visibility: public

Description: `IndexOf` searches for the pointer `Item` in the list of pointers, and returns the index of the pointer, if found.

If no pointer with the value `Item` was found, -1 is returned.

4.52.16 TFPList.IndexOfItem

Synopsis: Search an item in the list.

Declaration: `function IndexOfItem(Item: Pointer; Direction: TDirection) : Integer`

Visibility: public

Description: `IndexOfItem` has the same function as the IndexOf (272) function: it searches for `Item` in the list, and returns the index of the first found matching pointer. If none is found, -1 is returned.

In difference with the `IndexOf` function, it accepts a parameter `Direction` indicating the search direction: from the beginning of the list till the end of the list, or from the end of the list till the beginning. The `IndexOf` function starts at the beginning of the list. The search direction is only important if the item can appear multiple times in the list.

See also: TFPList.TDirection (**??**), TFPList.IndexOf (400)

4.52.17 TFPList.Insert

Synopsis: Inserts a new pointer in the list at a given position.

Declaration: `procedure Insert(Index: Integer; Item: Pointer)`

Visibility: public

Description: `Insert` inserts pointer `Item` at position `Index` in the list. All pointers starting from `Index` are shifted to the right.

If `Index` is not a valid position, then a EListError (275) exception is raised.

See also: Add (398), Delete (399)

4.52.18 TFPList.Last

Synopsis: Returns the last non-nil pointer in the list.

Declaration: `function Last : Pointer`

Visibility: public

Description: `Last` returns the value of the last non-nil pointer in the list.

If there are no pointers in the list or all pointers equal `Nil`, then `Nil` is returned.

See also: First (400)

4.52.19 TFPList.Move

Synopsis: Moves a pointer from one position in the list to another.

Declaration: `procedure Move(CurIndex: Integer; NewIndex: Integer)`

Visibility: public

Description: `Move` moves the pointer at position `CurIndex` to position `NewIndex`. This is done by storing the value at position `CurIndex`, deleting the pointer at position `CurIndex`, and reinserting the value at position `NewIndex`

If `CurIndex` or `Newindex` are not inside the valid range of indices, an EListError (275) exception is raised.

See also: Exchange (399)

4.52.20 TFPList.Assign

Synopsis: Assign performs the given operation on the list.

Declaration:
```
procedure Assign(ListA: TFPList; AOperator: TListAssignOp;
                 ListB: TFPList)
```

Visibility: public

Description: `Assign` can be used to merge or assign lists It is an extended version of the usual `TPersistent.Assign` mechanism. The arguments `ListA` and `ListB` are used as sources of pointers to add or remove elements from the current list, depending on the operation `AOperation`. The available operations are documented in the TListAssignOp (282) type.

See also: TFPList.Add (398), TFPList.Clear (398)

4.52.21 TFPList.Remove

Synopsis: Removes a value from the list.

Declaration:
```
function Remove(Item: Pointer) : Integer
```

Visibility: public

Description: `Remove` searches `Item` in the list, and, if it finds it, deletes the item from the list. Only the first occurrence of `Item` is removed.

See also: Delete (399), IndexOf (400), Insert (401)

4.52.22 TFPList.Pack

Synopsis: Removes `Nil` pointers from the list and frees unused memory.

Declaration:
```
procedure Pack
```

Visibility: public

Description: `Pack` removes all `nil` pointers from the list. The capacity of the list is then set to the number of pointers in the list. This method can be used to free unused memory if the list has grown to very large sizes and has a lot of unneeded nil pointers in it.

See also: TFPList.Clear (398)

4.52.23 TFPList.Sort

Synopsis: Sorts the pointers in the list.

Declaration:
```
procedure Sort(Compare: TListSortCompare)
```

Visibility: public

Description: `Sort>` sorts the pointers in the list. Two pointers are compared by passing them to the `Compare` function. The result of this function determines how the pointers will be sorted:

- If the result of this function is negative, the first pointer is assumed to be 'less' than the second and will be moved before the second in the list.
- If the function result is positive, the first pointer is assumed to be 'greater than' the second and will be moved after the second in the list.

- if the function result is zero, the pointers are assumed to be 'equal' and no moving will take place.

The sort is done using a quicksort algorithm.

4.52.24 TFPList.ForEachCall

Synopsis: Call a procedure or method for each pointer in the list.

Declaration:
```
procedure ForEachCall(proc2call: TListCallback; arg: pointer)
procedure ForEachCall(proc2call: TListStaticCallback; arg: pointer)
```

Visibility: public

Description: `ForEachCall` iterates over all pointers in the list and calls `proc2call`, passing it the pointer and the additional `arg` data pointer. `Proc2Call` can be a method or a static procedure.

Errors: None.

See also: TListStaticCallback (283), TListCallback (282)

4.52.25 TFPList.Capacity

Synopsis: Current capacity (i.e. number of pointers that can be stored) of the list.

Declaration:
```
Property Capacity : Integer
```

Visibility: public

Access: Read,Write

Description: `Capacity` contains the number of pointers the list can store before it starts to grow.

If a new pointer is added to the list using add (398) or insert (401), and there is not enough memory to store the new pointer, then the list will try to allocate more memory to store the new pointer. Since this is a time consuming operation, it is important that this operation be performed as little as possible. If it is known how many pointers there will be before filling the list, it is a good idea to set the capacity first before filling. This ensures that the list doesn't need to grow, and will speed up filling the list.

See also: SetCapacity (397), Count (403)

4.52.26 TFPList.Count

Synopsis: Current number of pointers in the list.

Declaration:
```
Property Count : Integer
```

Visibility: public

Access: Read,Write

Description: `Count` is the current number of (possibly `Nil`) pointers in the list. Since the list is zero-based, the index of the largest pointer is `Count-1`.

4.52.27 TFPList.Items

Synopsis: Provides access to the pointers in the list.

Declaration: `Property Items[Index: Integer]: Pointer; default`

Visibility: public

Access: Read,Write

Description: `Items` is used to access the pointers in the list. It is the default property of the `TFPList` class, so it can be omitted.

The list is zero-based, so Index must be in the range `0` to `Count-1`.

4.52.28 TFPList.List

Synopsis: Memory array where pointers are stored.

Declaration: `Property List : PPointerList`

Visibility: public

Access: Read

Description: `List` points to the memory space where the pointers are stored. This can be used to quickly copy the list of pointers to another location.

4.53 TFPListEnumerator

4.53.1 Description

`TFPListEnumerator` implements the #rtl.system.IEnumerator (1617) interface for the TFPList (397) class, so the `TFPList` class can be used in a `for ... in` loop. It is returned by the TFPList.GetEnumerator (400) method of `TFPList`.

See also: TFPList (397), TFPList.GetEnumerator (400), #rtl.system.IEnumerator (1617)

4.53.2 Method overview

Page	Method	Description
404	Create	Initialize a new instance of `TFPListEnumerator`.
405	GetCurrent	Return the current pointer in the list.
405	MoveNext	Move the position of the enumerator to the next position in the list.

4.53.3 Property overview

Page	Properties	Access	Description
405	Current	r	Current pointer in the list.

4.53.4 TFPListEnumerator.Create

Synopsis: Initialize a new instance of `TFPListEnumerator`.

Declaration: `constructor Create(AList: TFPList)`

Visibility: public

Description: `Create` initializes a new instance of `TFPListEnumerator` and keeps a reference to the list `AList` that will be enumerated.

See also: TFPList (397)

4.53.5 TFPListEnumerator.GetCurrent

Synopsis: Return the current pointer in the list.

Declaration: `function GetCurrent : Pointer`

Visibility: public

Description: `GetCurrent` returns the current pointer in the enumerator.

Errors: No checking is done on the validity of the current position.

See also: MoveNext (405)

4.53.6 TFPListEnumerator.MoveNext

Synopsis: Move the position of the enumerator to the next position in the list.

Declaration: `function MoveNext : Boolean`

Visibility: public

Description: `MoveNext` puts the pointer on the next item in the list, and returns `True` if this succeeded, or `False` if the pointer is past the last element in the list.

Errors: Note that if `False` is returned, calling `GetCurrent` will result in an exception.

See also: GetCurrent (405)

4.53.7 TFPListEnumerator.Current

Synopsis: Current pointer in the list.

Declaration: `Property Current : Pointer`

Visibility: public

Access: Read

Description: `Current` redefines GetCurrent (405) as a property.

See also: GetCurrent (405)

4.54 THandleStream

4.54.1 Description

`THandleStream` is an abstract descendant of the TStream (455) class that provides methods for a stream to handle all reading and writing to and from a handle, provided by the underlying OS. To this end, it overrides the Read (406) and Write (407) methods of TStream.

Remark

- `ThandleStream` does not obtain a handle from the OS by itself, it just handles reading and writing to such a handle by wrapping the system calls for reading and writing; Descendant classes should obtain a handle from the OS by themselves and pass it on in the inherited constructor.
- Contrary to Delphi, no seek is implemented for `THandleStream`, since pipes and sockets do not support this. The seek is implemented in descendant methods that support it.

See also: TStream (455), TFileStream (395)

4.54.2 Method overview

Page	Method	Description
406	Create	Create a handlestream from an OS Handle.
406	Read	Overrides standard read method.
407	Seek	Overrides the Seek method.
407	Write	Overrides standard write method.

4.54.3 Property overview

Page	Properties	Access	Description
407	Handle	r	The OS handle of the stream.

4.54.4 THandleStream.Create

Synopsis: Create a handlestream from an OS Handle.

Declaration: `constructor Create(AHandle: THandle)`

Visibility: public

Description: `Create` creates a new instance of a `THandleStream` class. It stores `AHandle` in an internal variable and then calls the inherited constructor.

See also: TStream (455)

4.54.5 THandleStream.Read

Synopsis: Overrides standard read method.

Declaration: `function Read(var Buffer; Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Read` overrides the Read (456) method of `TStream`. It uses the Handle (407) property to read the `Count` bytes into `Buffer`

If no error occurs while reading, the number of bytes actually read will be returned.

Errors: If the operating system reports an error while reading from the handle, -1 is returned.

See also: TStream.Read (456), THandleStream.Write (407), THandleStream.Handle (407)

4.54.6 THandleStream.Write

Synopsis: Overrides standard write method.

Declaration:
```
function Write(const Buffer; Count: LongInt) : LongInt;  Override
```

Visibility: public

Description: `Write` overrides the Write (457) method of `TStream`. It uses the Handle (407) property to write the `Count` bytes from `Buffer`.

If no error occurs while writing, the number of bytes actually written will be returned.

Errors: If the operating system reports an error while writing to the handle, 0 is returned.

See also: TStream.Read (456), THandleStream.Write (407), THandleStream.Handle (407)

4.54.7 THandleStream.Seek

Synopsis: Overrides the Seek method.

Declaration:
```
function Seek(const Offset: Int64; Origin: TSeekOrigin) : Int64
                  ;  Override
```

Visibility: public

Description: `seek` uses the FileSeek (1723) method to position the stream on the desired position. Note that handle stream descendants (notably pipes) can override the method to prevent the seek.

4.54.8 THandleStream.Handle

Synopsis: The OS handle of the stream.

Declaration:
```
Property Handle : THandle
```

Visibility: public

Access: Read

Description: `Handle` represents the Operating system handle to which reading and writing is done. The handle can be read only, i.e. it cannot be set after the `THandlestream` instance was created. It should be passed to the constructor THandleStream.Create (406)

See also: THandleStream (406), THandleStream.Create (406)

4.55 TInterfacedPersistent

4.55.1 Description

`TInterfacedPersistent` is a direct descendant of TPersistent (435) which implements the #rtl.system.IInterface (1399) interface. In particular, it implements the `QueryInterface` as a public method.

See also: IInterface (1399)

4.55.2 Interfaces overview

Page	Interfaces	Description
1399	IInterface	Basic interface for all COM based interfaces.

4.55.3 Method overview

Page	Method	Description
408	AfterConstruction	Overrides the standard `AfterConstruction` method.
408	QueryInterface	Implementation of `IInterface.QueryInterface`.

4.55.4 TInterfacedPersistent.QueryInterface

Synopsis: Implementation of `IInterface.QueryInterface`.

Declaration: `function QueryInterface(const IID: TGuid; out Obj) : HRESULT;  Virtual`

Visibility: public

Description: `QueryInterface` simply calls `GetInterface` using the specified `IID`, and returns the correct values.

See also: TObject.GetInterface (1631)

4.55.5 TInterfacedPersistent.AfterConstruction

Synopsis: Overrides the standard `AfterConstruction` method.

Declaration: `procedure AfterConstruction;  Override`

Visibility: public

Description: `AfterConstruction` is overridden to do some extra interface housekeeping: a reference to the `IInterface` interface of the owning class is obtained (if it exists).

4.56 TInterfaceList

4.56.1 Description

`TInterfaceList` is a standard implementation of the IInterfaceList (316) interface. It uses a TThreadList (522) instance to store the list of interfaces.

See also: IInterfaceList (316), TList (415)

4.56.2 Interfaces overview

Page	Interfaces	Description
316	IInterfaceList	Interface for maintaining a list of interfaces.

4.56.3 Method overview

Page	Method	Description
411	Add	Add an interface to the list.
410	Clear	Removes all interfaces from the list.
409	Create	Create a new instance of `TInterfaceList`.
410	Delete	Delete an interface from the list.
409	Destroy	Destroys the list of interfaces.
410	Exchange	Exchange 2 interfaces in the list.
413	Expand	Expands the list.
410	First	Returns the first non-`Nil` element in the list.
411	GetEnumerator	Create an `IEnumerator` instance.
411	IndexOf	Returns the index of an interface.
411	Insert	Insert an interface to the list.
412	Last	Returns the last non-`Nil` element in the list.
412	Lock	Lock the list.
412	Remove	Remove an interface from the list.
412	Unlock	UnLocks a locked list.

4.56.4 Property overview

Page	Properties	Access	Description
413	Capacity	rw	The current capacity of the list.
413	Count	rw	The current number of elements in the list.
413	Items	rw	Array-based access to the list's items.

4.56.5 TInterfaceList.Create

Synopsis: Create a new instance of `TInterfaceList`.

Declaration:
```
constructor Create
```

Visibility: public

Description: `Create` creates a new instance of the `TInterfaceList` class. It sets up the internal structures needed to store the list of interfaces.

See also: Destroy (409)

4.56.6 TInterfaceList.Destroy

Synopsis: Destroys the list of interfaces.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` first calls Clear (410) and then frees the `TInterfaceList` instance from memory.

Note that the `Clear` method decreases the reference count of all interfaces.

See also: Create (409), Clear (410)

4.56.7 TInterfaceList.Clear

Synopsis: Removes all interfaces from the list.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` is the implementation of the IInterfaceList.Clear (319) method. It removes all interfaces from the list. It does this by setting each element in the list to `Nil`, in this way the reference count of each interface in the list is decreased.

See also: IInterfaceList.Clear (319), Add (411), Destroy (409), TList.Clear (418), TFPList.Clear (398)

4.56.8 TInterfaceList.Delete

Synopsis: Delete an interface from the list.

Declaration: `procedure Delete(index: Integer)`

Visibility: public

Description: `Delete` is the implementation of the IInterfaceList.Delete (319) method. It clears the slot first and then removes the element from the list.

See also: IInterfaceList.Delete (319), TInterfaceList.Remove (412), TInterfaceList.Add (411), TList.Delete (418), TFPList.Delete (399)

4.56.9 TInterfaceList.Exchange

Synopsis: Exchange 2 interfaces in the list.

Declaration: `procedure Exchange(index1: Integer; index2: Integer)`

Visibility: public

Description: `Exchange` is the implementation of the IInterfaceList.Exchange (319) method. It exchanges the position of 2 interfaces in the list.

See also: IInterfaceList.Exchange (319), TInterfaceList.Delete (410), TInterfaceList.Add (411), TList.Exchange (419), TFPList.Exchange (399)

4.56.10 TInterfaceList.First

Synopsis: Returns the first non-`Nil` element in the list.

Declaration: `function First : IUnknown`

Visibility: public

Description: `First` is the implementation of the IInterfaceList.First (319) method. It returns the first non-`Nil` element from the list.

See also: IInterfaceList.First (319), TList.First (420)

4.56.11 TInterfaceList.GetEnumerator

Synopsis: Create an `IEnumerator` instance.

Declaration: `function GetEnumerator : TInterfaceListEnumerator`

Visibility: public

Description: `GetEnumerator` is the implementation of the IEnumerable (1616) interface for `TInterfaceList`. It creates a TInterfaceListEnumerator (414) instance and returns it's IEnumerator (1617) interface. The enumerator enumerates all interfaces in the list.

See also: TInterfaceListEnumerator (414), IEnumerator (1617), IEnumerable (1616)

4.56.12 TInterfaceList.IndexOf

Synopsis: Returns the index of an interface.

Declaration: `function IndexOf(const item: IUnknown) : Integer`

Visibility: public

Description: `IndexOf` is the implementation of the IInterfaceList.IndexOf (320) method. It returns the zero-based index in the list of the indicated interface, or -1 if the index is not in the list.

See also: IInterfaceList.IndexOf (320), TList.IndexOf (420)

4.56.13 TInterfaceList.Add

Synopsis: Add an interface to the list.

Declaration: `function Add(item: IUnknown) : Integer`

Visibility: public

Description: `Add` is the implementation of the IInterfaceList.Add (320) method. It adds an interface to the list, and returns the location of the new element in the list. This operation will increment the reference count of the interface.

See also: IInterfaceList.Add (320), TInterfaceList.Delete (410), TInterfaceList.Insert (411), TList.Add (418), TFPList.Add (398)

4.56.14 TInterfaceList.Insert

Synopsis: Insert an interface to the list.

Declaration: `procedure Insert(i: Integer; item: IUnknown)`

Visibility: public

Description: `Insert` is the implementation of the IInterfaceList.Insert (320) method. It inserts an interface in the list at the indicated position. This operation will increment the reference count of the interface.

See also: IInterfaceList.Insert (320), TInterfaceList.Delete (410), TInterfaceList.Add (411), TList.Insert (420), TFPList.Insert (401)

4.56.15 TInterfaceList.Last

Synopsis: Returns the last non-`Nil` element in the list.

Declaration: `function Last : IUnknown`

Visibility: public

Description: `Last` is the implementation of the IInterfaceList.Last (320) method. It returns the last non-`Nil` element from the list.

See also: IInterfaceList.Last (320), TInterfaceList.First (410), TList.Last (421), TFPList.Last (401)

4.56.16 TInterfaceList.Remove

Synopsis: Remove an interface from the list.

Declaration: `function Remove(item: IUnknown) : Integer`

Visibility: public

Description: `Remove` is the implementation of the IInterfaceList.Remove (320) method. It removes the first occurrence of the interface from the list.

See also: IInterfaceList.Remove (320), TInterfaceList.Delete (410), TInterfaceList.IndexOf (411), TList.Remove (421), TFPList.Remove (402)

4.56.17 TInterfaceList.Lock

Synopsis: Lock the list.

Declaration: `procedure Lock`

Visibility: public

Description: `Lock` locks the list. It is the implementation of the IInterfaceList.Lock (321) method. It limits access to the list to the current thread.

See also: IInterfaceList.Lock (321), TInterfaceList.UnLock (412), TThreadList.LockList (523)

4.56.18 TInterfaceList.Unlock

Synopsis: UnLocks a locked list.

Declaration: `procedure Unlock`

Visibility: public

Description: `UnLock` unlocks the list. It is the implementation of the IInterfaceList.UnLock (321) method. After a call to unlock, the current thread releases the list for manipulation by other threads.

See also: IInterfaceList.UnLock (321), TInterfaceList.Lock (412), TThreadList.UnLockList (524)

4.56.19 TInterfaceList.Expand

Synopsis: Expands the list.

Declaration: `function Expand : TInterfaceList`

Visibility: public

Description: `Expand` calls the expand method from the internally used list. It returns itself.

See also: TList.Expand (419)

4.56.20 TInterfaceList.Capacity

Synopsis: The current capacity of the list.

Declaration: `Property Capacity : Integer`

Visibility: public

Access: Read,Write

Description: `Capacity` is the number of elements that the list can contain without needing to allocate more memory.

See also: IInterfaceList.Capacity (321), TInterfaceList.Count (413), TList.Capacity (422), TFPList.Capacity (403)

4.56.21 TInterfaceList.Count

Synopsis: The current number of elements in the list.

Declaration: `Property Count : Integer`

Visibility: public

Access: Read,Write

Description: `Count` is the number of elements in the list. This can include `Nil` elements. Note that the elements are zero-based, and thus are indexed from 0 to `Count-1`.

See also: IInterfaceList.Count (321), TInterfaceList.Items (413), TInterfaceList.Capacity (413), TList.Count (423), TFPList.Count (403)

4.56.22 TInterfaceList.Items

Synopsis: Array-based access to the list's items.

Declaration: `Property Items[Index: Integer]: IUnknown; default`

Visibility: public

Access: Read,Write

Description: `Items` provides indexed access to the elements in the list. Note that the elements are zero-based, and thus are indexed from 0 to `Count-1`. The items are read-write. It is not possible to add elements to the list by accessing an element with index larger or equal to Count (413).

See also: IInterfaceList.Items (322), TInterfaceList.Count (413), TList.Items (423), TFPList.Items (404)

4.57 TInterfaceListEnumerator

4.57.1 Description

`TInterfaceListEnumerator` implements the #rtl.system.IEnumerator (1617) interface for the TInterfaceList (408) class, so the `TInterfaceList` class can be used in a `for ... in` loop over the TInterfaceList.Components (408) child components of the component. It is returned by the TInterfaceList.GetEnumerator (411) method of `TInterfaceList`.

See also: TInterfaceList (408), TInterfaceList.GetEnumerator (411), #rtl.system.IEnumerator (1617)

4.57.2 Method overview

Page	Method	Description
414	Create	Initialize a new instance of `TInterfaceListEnumerator`.
414	GetCurrent	Return the current pointer in the list.
415	MoveNext	Move the position of the enumerator to the next position in the children of the component.

4.57.3 Property overview

Page	Properties	Access	Description
415	Current	r	Current pointer in the list.

4.57.4 TInterfaceListEnumerator.Create

Synopsis: Initialize a new instance of `TInterfaceListEnumerator`.

Declaration: `constructor Create(AList: TInterfaceList)`

Visibility: public

Description: `Create` initializes a new instance of `TInterfaceListEnumerator` and keeps a reference to the component `AComponent` that will be enumerated.

See also: TInterfaceList (408)

4.57.5 TInterfaceListEnumerator.GetCurrent

Synopsis: Return the current pointer in the list.

Declaration: `function GetCurrent : IUnknown`

Visibility: public

Description: `GetCurrent` returns the current interface in the TInterfaceList (408) list.

Errors: No checking is done on the validity of the current position.

See also: MoveNext (415), TInterfaceList.Components (408)

4.57.6 TInterfaceListEnumerator.MoveNext

Synopsis: Move the position of the enumerator to the next position in the children of the component.

Declaration: `function MoveNext : Boolean`

Visibility: public

Description: `MoveNext` puts the pointer on the next interface in the list, and returns `True` if this succeeded, or `False` if the pointer is past the last interface in the list.

Errors: Note that if `False` is returned, calling `GetCurrent` will result in an exception.

See also: GetCurrent (414)

4.57.7 TInterfaceListEnumerator.Current

Synopsis: Current pointer in the list.

Declaration: `Property Current : IUnknown`

Visibility: public

Access: Read

Description: `Current` redefines GetCurrent (414) as a property.

See also: GetCurrent (414)

4.58 TList

4.58.1 Description

`TList` is a class that can be used to manage collections of pointers. It introduces methods and properties to store the pointers, search in the list of pointers, sort them. It manages its memory by itself, no intervention for that is needed. It has an event notification mechanism which allows to notify of list changes. This slows down some of `TList` mechanisms, and if no notification is used, TFPList (397) may be used instead.

To manage collections of strings, it is better to use a TStrings (475) descendant such as TStringList (470). To manage general objects, a TCollection (365) class exists, from which a descendant can be made to manage collections of various kinds.

See also: TStrings (475), TCollection (365)

4.58.2 Interfaces overview

Page	Interfaces	Description
314	IFPObserved	Interface implemented by an object that can be observed.

4.58.3 Method overview

Page	Method	Description
418	Add	Adds a new pointer to the list.
418	AddList	Add all pointers from another list.
421	Assign	Copy the contents of other lists.
418	Clear	Clears the pointer list.
416	Create	Class to manage collections of pointers.
418	Delete	Removes a pointer from the list.
416	Destroy	Destroys the list and releases the memory used to store the list elements.
419	Error	Raises an EListError (275) exception.
419	Exchange	Exchanges two pointers in the list.
419	Expand	Increases the capacity of the list if needed.
419	Extract	Remove the first occurrence of a pointer from the list.
420	First	Returns the first non-nil pointer in the list.
417	FPOAttachObserver	Add an observer to the list of observers.
417	FPODetachObserver	Remove an observer from the list of observers.
417	FPONotifyObservers	Notify observers of changes in the list.
420	GetEnumerator	Create an `IEnumerator` instance.
420	IndexOf	Returns the index of a given pointer.
420	Insert	Inserts a new pointer in the list at a given position.
421	Last	Returns the last non-nil pointer in the list.
421	Move	Moves a pointer from one position in the list to another.
422	Pack	Removes `Nil` pointers from the list and frees unused memory.
421	Remove	Removes a value from the list.
422	Sort	Sorts the pointers in the list.

4.58.4 Property overview

Page	Properties	Access	Description
422	Capacity	rw	Current capacity (i.e. number of pointers that can be stored) of the list.
423	Count	rw	Current number of pointers in the list.
423	Items	rw	Provides access to the pointers in the list.
423	List	r	Memory array where pointers are stored.

4.58.5 TList.Create

Synopsis: Class to manage collections of pointers.

Declaration: `constructor Create`

Visibility: public

Description: `TList.Create` creates a new instance of `TList`. It clears the list and prepares it for use.

See also: TList (415), TList.Destroy (416)

4.58.6 TList.Destroy

Synopsis: Destroys the list and releases the memory used to store the list elements.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` destroys the list and releases the memory used to store the list elements. The elements themselves are in no way touched, i.e. any memory they point to must be explicitly released before calling the destructor.

4.58.7 TList.FPOAttachObserver

Synopsis: Add an observer to the list of observers.

Declaration:
```
procedure FPOAttachObserver(AObserver: TObject)
```

Visibility: public

Description: `FPOAttachObserver` is part of the implementation of the IFPObserved (314) interface in `Tlist`. It adds a new observer to the list of observers. Calling this multiple times will add the observed object multiple times to the list.

Errors: An `EObserver` exception may be raised if `AObject` does not implement the IFPObserver (315) interface.

See also: IFPObserver (315), IFPObserved.FPOAttachObserver (314), IFPObserved (314)

4.58.8 TList.FPODetachObserver

Synopsis: Remove an observer from the list of observers.

Declaration:
```
procedure FPODetachObserver(AObserver: TObject)
```

Visibility: public

Description: `FPODetachObserver` is part of the implementation of the IFPObserved (314) interface in `Tlist`. It removes the first found instance of the observer from the list of observers.

See also: IFPObserved (314), IFPObserved.FPODetachObserver (315), IFPObserver (315)

4.58.9 TList.FPONotifyObservers

Synopsis: Notify observers of changes in the list.

Declaration:
```
procedure FPONotifyObservers(ASender: TObject;
                             AOperation: TFPObservedOperation;
                             Data: Pointer)
```

Visibility: public

Description: `FPONotifyObservers` is called to notify observers of changes in the list. The following notifications are sent:

ooAddItemwhen a pointer is added. `Data` is the pointer that is added.

ooDeleteItemwhen a pointer is deleted or extracted. `Data` is the pointer that is deleted or extracted.

ooChangecalled when 2 pointers are exchanged.

ooFreeCalled when the list is freed.

See also: FPODetachObserver (272), FPOAttachObserver (272), Add (272), Exchange (272), Delete (272), Extract (272)

4.58.10 TList.AddList

Synopsis: Add all pointers from another list.

Declaration: `procedure AddList(AList: TList)`

Visibility: public

Description: `AddList` adds all pointers from `AList` to the list. If a pointer is already present, it is added a second time.

See also: TList.Assign (421), TFPList.AddList (398)

4.58.11 TList.Add

Synopsis: Adds a new pointer to the list.

Declaration: `function Add(Item: Pointer) : Integer`

Visibility: public

Description: `Add` adds a new pointer to the list after the last pointer (i.e. at position `Count`, thus increasing the item count with 1. If the list is at full capacity, the capacity of the list is expanded, using the Grow (415) method.

To insert a pointer at a certain position in the list, use the Insert (420) method instead.

See also: Delete (418), Grow (415), Insert (420)

4.58.12 TList.Clear

Synopsis: Clears the pointer list.

Declaration: `procedure Clear;  Virtual`

Visibility: public

Description: `Clear` removes all pointers from the list, and sets the capacity to 0, thus freeing any memory allocated to maintain the list.

See also: Destroy (416)

4.58.13 TList.Delete

Synopsis: Removes a pointer from the list.

Declaration: `procedure Delete(Index: Integer)`

Visibility: public

Description: `Delete` removes the pointer at position `Index` from the list, shifting all following pointers one position up (or to the left).

The memory the pointer is pointing to is *not* deallocated.

4.58.14 TList.Error

Synopsis: Raises an EListError (275) exception.

Declaration: `class procedure Error(const Msg: string; Data: PtrInt); Virtual`

Visibility: public

Description: `Error` raises an EListError (275) exception, with a message formatted with `Msg` and `Data`.

4.58.15 TList.Exchange

Synopsis: Exchanges two pointers in the list.

Declaration: `procedure Exchange(Index1: Integer; Index2: Integer)`

Visibility: public

Description: `Exchange` exchanges the pointers at positions `Index1` and `Index2`. Both pointers must be withing the current range of the list, or an EListError (275) exception will be raised.

4.58.16 TList.Expand

Synopsis: Increases the capacity of the list if needed.

Declaration: `function Expand : TList`

Visibility: public

Description: `Expand` increases the capacity of the list if the current element count matches the current list capacity.

The capacity is increased according to the following algorithm:

1. If the capacity is less than 3, the capacity is increased with 4.
2. If the capacity is larger than 3 and less than 8, the capacity is increased with 8.
3. If the capacity is larger than 8, the capacity is increased with 16.

The return value is `Self`.

See also: Capacity (422)

4.58.17 TList.Extract

Synopsis: Remove the first occurrence of a pointer from the list.

Declaration: `function Extract(item: Pointer) : Pointer`

Visibility: public

Description: `Extract` searched for an occurrence of `item`, and if a match is found, the match is deleted from the list. If no match is found, nothing is deleted. If `Item` was found, the result is `Item`. If `Item` was not found, the result is `Nil`. A `lnExtracted` notification event is triggered if an element is extracted from the list.

See also: TList.Delete (418), TList.IndexOf (420), TList.Remove (421)

4.58.18 TList.First

Synopsis: Returns the first non-nil pointer in the list.

Declaration: `function First : Pointer`

Visibility: public

Description: `First` returns the value of the first non-nil pointer in the list.

If there are no pointers in the list or all pointers equal `Nil`, then `Nil` is returned.

See also: Last (421)

4.58.19 TList.GetEnumerator

Synopsis: Create an `IEnumerator` instance.

Declaration: `function GetEnumerator : TListEnumerator`

Visibility: public

Description: `GetEnumerator` is the implementation of the IEnumerable (1616) interface for `TList`. It creates a TListEnumerator (423) instance and returns it's IEnumerator (1617) interface.

See also: TListEnumerator (423), IEnumerator (1617), IEnumerable (1616)

4.58.20 TList.IndexOf

Synopsis: Returns the index of a given pointer.

Declaration: `function IndexOf(Item: Pointer) : Integer`

Visibility: public

Description: `IndexOf` searches for the pointer `Item` in the list of pointers, and returns the index of the pointer, if found.

If no pointer with the value `Item` was found, -1 is returned.

4.58.21 TList.Insert

Synopsis: Inserts a new pointer in the list at a given position.

Declaration: `procedure Insert(Index: Integer; Item: Pointer)`

Visibility: public

Description: `Insert` inserts pointer `Item` at position `Index` in the list. All pointers starting from `Index` are shifted to the right.

If `Index` is not a valid position, then a EListError (275) exception is raised.

See also: Add (418), Delete (418)

4.58.22 TList.Last

Synopsis: Returns the last non-nil pointer in the list.

Declaration: `function Last : Pointer`

Visibility: public

Description: `Last` returns the value of the last non-nil pointer in the list.

If there are no pointers in the list or all pointers equal `Nil`, then `Nil` is returned.

See also: First (420)

4.58.23 TList.Move

Synopsis: Moves a pointer from one position in the list to another.

Declaration: `procedure Move(CurIndex: Integer; NewIndex: Integer)`

Visibility: public

Description: `Move` moves the pointer at position `CurIndex` to position `NewIndex`. This is done by storing the value at position `CurIndex`, deleting the pointer at position `CurIndex`, and reinserting the value at position `NewIndex`

If `CurIndex` or `Newindex` are not inside the valid range of indices, an EListError (275) exception is raised.

See also: Exchange (419)

4.58.24 TList.Assign

Synopsis: Copy the contents of other lists.

Declaration: `procedure Assign(ListA: TList; AOperator: TListAssignOp; ListB: TList)`

Visibility: public

Description: `Assign` can be used to merge or assign lists It is an extended version of the usual `TPersistent.Assign` mechanism. The arguments `ListA` and `ListB` are used as sources of pointers to add or remove elements from the current list, depending on the operation `AOperation`. The available operations are documented in the TListAssignOp (282) type.

See also: TList.Clear (418)

4.58.25 TList.Remove

Synopsis: Removes a value from the list.

Declaration: `function Remove(Item: Pointer) : Integer`

Visibility: public

Description: `Remove` searches `Item` in the list, and, if it finds it, deletes the item from the list. Only the first occurrence of `Item` is removed.

See also: Delete (418), IndexOf (420), Insert (420)

4.58.26 TList.Pack

Synopsis: Removes `Nil` pointers from the list and frees unused memory.

Declaration: `procedure Pack`

Visibility: public

Description: `Pack` removes all `nil` pointers from the list. The capacity of the list is then set to the number of pointers in the list. This method can be used to free unused memory if the list has grown to very large sizes and has a lot of unneeded nil pointers in it.

See also: TList.Clear (418)

4.58.27 TList.Sort

Synopsis: Sorts the pointers in the list.

Declaration: `procedure Sort(Compare: TListSortCompare)`

Visibility: public

Description: `Sort>` sorts the pointers in the list. Two pointers are compared by passing them to the `Compare` function. The result of this function determines how the pointers will be sorted:

- If the result of this function is negative, the first pointer is assumed to be 'less' than the second and will be moved before the second in the list.
- If the function result is positive, the first pointer is assumed to be 'greater than' the second and will be moved after the second in the list.
- if the function result is zero, the pointers are assumed to be 'equal' and no moving will take place.

The sort is done using a quicksort algorithm.

4.58.28 TList.Capacity

Synopsis: Current capacity (i.e. number of pointers that can be stored) of the list.

Declaration: `Property Capacity : Integer`

Visibility: public

Access: Read,Write

Description: `Capacity` contains the number of pointers the list can store before it starts to grow.

If a new pointer is added to the list using add (418) or insert (420), and there is not enough memory to store the new pointer, then the list will try to allocate more memory to store the new pointer. Since this is a time consuming operation, it is important that this operation be performed as little as possible. If it is known how many pointers there will be before filling the list, it is a good idea to set the capacity first before filling. This ensures that the list doesn't need to grow, and will speed up filling the list.

See also: SetCapacity (415), Count (423)

4.58.29 TList.Count

Synopsis: Current number of pointers in the list.

Declaration: `Property Count : Integer`

Visibility: public

Access: Read,Write

Description: `Count` is the current number of (possibly `Nil`) pointers in the list. Since the list is zero-based, the index of the largest pointer is `Count-1`.

4.58.30 TList.Items

Synopsis: Provides access to the pointers in the list.

Declaration: `Property Items[Index: Integer]: Pointer; default`

Visibility: public

Access: Read,Write

Description: `Items` is used to access the pointers in the list. It is the default property of the `TList` class, so it can be omitted.

The list is zero-based, so Index must be in the range `0` to `Count-1`.

4.58.31 TList.List

Synopsis: Memory array where pointers are stored.

Declaration: `Property List : PPointerList`

Visibility: public

Access: Read

Description: `List` points to the memory space where the pointers are stored. This can be used to quickly copy the list of pointers to another location.

4.59 TListEnumerator

4.59.1 Description

`TListEnumerator` implements the #rtl.system.IEnumerator (1617) interface for the TList (415) class, so the `TList` class can be used in a `for ... in` loop. It is returned by the TList.GetEnumerator (420) method of `TList`.

See also: TList (415), TList.GetEnumerator (420), #rtl.system.IEnumerator (1617)

4.59.2 Method overview

Page	Method	Description
424	Create	Initialize a new instance of `TListEnumerator`.
424	GetCurrent	Return the current pointer in the list.
424	MoveNext	Move the position of the enumerator to the next position in the list.

4.59.3 Property overview

Page	Properties	Access	Description
424	Current	r	Current pointer in the list.

4.59.4 TListEnumerator.Create

Synopsis: Initialize a new instance of `TListEnumerator`.

Declaration: `constructor Create(AList: TList)`

Visibility: public

Description: `Create` initializes a new instance of `TListEnumerator` and keeps a reference to the list `AList` that will be enumerated.

See also: TList (415)

4.59.5 TListEnumerator.GetCurrent

Synopsis: Return the current pointer in the list.

Declaration: `function GetCurrent : Pointer`

Visibility: public

Description: `GetCurrent` returns the current pointer in the enumerator.

Errors: No checking is done on the validity of the current position.

See also: MoveNext (424)

4.59.6 TListEnumerator.MoveNext

Synopsis: Move the position of the enumerator to the next position in the list.

Declaration: `function MoveNext : Boolean`

Visibility: public

Description: `MoveNext` puts the pointer on the next item in the list, and returns `True` if this succeeded, or `False` if the pointer is past the last element in the list.

Errors: Note that if `False` is returned, calling `GetCurrent` will result in an exception.

See also: GetCurrent (424)

4.59.7 TListEnumerator.Current

Synopsis: Current pointer in the list.

Declaration: `Property Current : Pointer`

Visibility: public

Access: Read

Description: `Current` redefines GetCurrent (424) as a property.

See also: GetCurrent (424)

4.60 TMemoryStream

4.60.1 Description

`TMemoryStream` is a TStream (455) descendant that stores it's data in memory. It descends directly from TCustomMemoryStream (387) and implements the necessary to allocate and de-allocate memory directly from the heap. It implements the Write (426) method which is missing in `TCustomMemoryStream`.

`TMemoryStream` also introduces methods to load the contents of another stream or a file into the memory stream.

It is not necessary to do any memory management manually, as the stream will allocate or de-allocate memory as needed. When the stream is freed, all allocated memory will be freed as well.

See also: TCustomMemoryStream (387), TStream (455)

4.60.2 Method overview

Page	Method	Description
425	Clear	Zeroes the position, capacity and size of the stream.
425	Destroy	Frees any allocated memory and destroys the memory stream.
426	LoadFromFile	Loads the contents of a file into memory.
425	LoadFromStream	Loads the contents of a stream into memory.
426	SetSize	Sets the size for the memory stream.
426	Write	Writes data to the stream's memory.

4.60.3 TMemoryStream.Destroy

Synopsis: Frees any allocated memory and destroys the memory stream.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` clears the memory stream, thus in effect freeing any memory allocated for it, and then frees the memory stream.

4.60.4 TMemoryStream.Clear

Synopsis: Zeroes the position, capacity and size of the stream.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` sets the position and size to 0, and sets the capacity of the stream to 0, thus freeing all memory allocated for the stream.

See also: TStream.Size (465), TStream.Position (464), TCustomMemoryStream.Memory (389)

4.60.5 TMemoryStream.LoadFromStream

Synopsis: Loads the contents of a stream into memory.

Declaration: `procedure LoadFromStream(Stream: TStream)`

Visibility: public

Description: `LoadFromStream` loads the contents of `Stream` into the memorybuffer of the stream. Any previous contents of the memory stream are overwritten. Memory is allocated as needed.

Remark The `LoadFromStream` uses the Size (465) property of `Stream` to determine how much memory must be allocated. Some streams do not allow the stream size to be determined, so care must be taken when using this method.

This method will work much faster than the use of the TStream.CopyFrom (459) method:

```
Seek(0,soFromBeginning);
  CopyFrom(Stream,Stream.Size);
```

because the `CopyFrom` method copies the contents in blocks, while `LoadFromStream` reads the contents of the stream as one big block.

Errors: If an error occurs when reading from the stream, an EStreamError (312) may occur.

See also: TStream.CopyFrom (459), TMemoryStream.LoadFromFile (426)

4.60.6 TMemoryStream.LoadFromFile

Synopsis: Loads the contents of a file into memory.

Declaration: `procedure LoadFromFile(const FileName: string)`

Visibility: public

Description: `LoadFromFile` loads the contents of the file with name `FileName` into the memory stream. The current contents of the memory stream is replaced by the contents of the file. Memory is allocated as needed.

The `LoadFromFile` method simply creates a filestream and then calls the TMemoryStream.LoadFromStream (425) method.

See also: TMemoryStream.LoadFromStream (425)

4.60.7 TMemoryStream.SetSize

Synopsis: Sets the size for the memory stream.

Declaration: `procedure SetSize(const NewSize: Int64);  Override`

Visibility: public

Description: `SetSize` sets the size of the memory stream to `NewSize`. This will set the capacity of the stream to `NewSize` and correct the current position in the stream when needed.

See also: TStream.Position (464), TStream.Size (465)

4.60.8 TMemoryStream.Write

Synopsis: Writes data to the stream's memory.

Declaration: `function Write(const Buffer; Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Write` writes `Count` bytes from `Buffer` to the stream's memory, starting at the current position in the stream. If more memory is needed than currently allocated, more memory will be allocated. Any contents in the memory stream at the current position will be overwritten. The function returns the number of bytes actually written (which should under normal circumstances always equal `Count`).

This method overrides the TStream.Write (457) method.

Errors: If no more memory could be allocated, then an exception will be raised.

See also: TCustomMemoryStream.Read (388)

4.61 TOwnedCollection

4.61.1 Description

`TOwnedCollection` automatically maintains owner information, so it can be displayed in an IDE. Collections that should be displayed in an IDE should descend from `TOwnedCollection` or must implement a `GetOwner` function.

See also: TCollection (365)

4.61.2 Method overview

Page	Method	Description
427	Create	Create a new `TOwnerCollection` instance.

4.61.3 TOwnedCollection.Create

Synopsis: Create a new `TOwnerCollection` instance.

Declaration:
```
constructor Create(AOwner: TPersistent;
                   AItemClass: TCollectionItemClass)
```

Visibility: public

Description: `Create` creates a new instance of `TOwnedCollection` and stores the `AOwner` references. It will the value returned in the TCollection.Owner (367) property of the collection. The `ItemClass` class reference is passed on to the inherited constructor, and will be used to create new instances in the Insert (369) and Add (367) methods.

See also: TCollection.Create (366), TCollection.Owner (367)

4.62 TOwnerStream

4.62.1 Description

`TOwnerStream` can be used when creating stream chains such as when using encryption and compression streams. It keeps a reference to the source stream and will automatically free the source stream when ready (if the SourceOwner (429) property is set to `True`).

See also: TStream (455), TOwnerStream.Source (428), TOwnerStream.SourceOwner (429)

4.62.2 Method overview

Page	Method	Description
428	Create	Create a new instance of `TOwnerStream`.
428	Destroy	Destroys the `TOwnerStream` instance and the source stream.

4.62.3 Property overview

Page	Properties	Access	Description
428	Source	r	Reference to the source stream.
429	SourceOwner	rw	Indicates whether the ownerstream owns it's source.

4.62.4 TOwnerStream.Create

Synopsis: Create a new instance of `TOwnerStream`.

Declaration: `constructor Create(ASource: TStream)`

Visibility: public

Description: `Create` instantiates a new instance of `TOwnerStream` and stores the reference to `AStream`. If `SourceOwner` is `True`, the source stream will also be freed when the instance is destroyed.

See also: TOwnerStream.Destroy (428), TOwnerStream.Source (428), TOwnerStream.SourceOwner (429)

4.62.5 TOwnerStream.Destroy

Synopsis: Destroys the `TOwnerStream` instance and the source stream.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` frees the source stream if the `SourceOwner` property is `True`.

See also: TOwnerStream.Create (428), TOwnerStream.Source (428), TOwnerStream.SourceOwner (429)

4.62.6 TOwnerStream.Source

Synopsis: Reference to the source stream.

Declaration: `Property Source : TStream`

Visibility: public

Access: Read

Description: `Source` is the source stream. It should be used by descendant streams to access the source stream to read from or write to.

Do not free the `Source` reference directly if `SourceOwner` is `True`. In that case the owner stream instance will free the source stream itself.

See also: TOwnerStream.Create (428)

4.62.7 TOwnerStream.SourceOwner

Synopsis: Indicates whether the ownerstream owns it's source.

Declaration: `Property SourceOwner : Boolean`

Visibility: public

Access: Read,Write

Description: `SourceOwner` indicates whether the `TOwnerStream` owns it's `Source` stream or not. If this property is `True` then the `Source` stream is freed when the `TOwnerStream` instance is freed.

See also: TOwnerStream.Source (428), TOwnerStream.Destroy (428)

4.63 TParser

4.63.1 Description

This class breaks a stream of text data in tokens. Its primary use is to help reading the contents of a form file (usually a file with dfm, xfm or lfm extension), and for this reason it isn't suitable to be used as a general parser.

The parser is always positioned on a certain token, whose type is stored in the Token (435) property. Various methods are provided to obtain the token value in the desired format.

To advance to the next token, invoke NextToken (432) method.

See also: TParser.Token (435), TParser.NextToken (432)

4.63.2 Method overview

Page	Method	Description
430	CheckToken	Checks whether the token if of the given type.
430	CheckTokenSymbol	Checks whether the token equals the given symbol.
430	Create	Creates a new parser instance.
430	Destroy	Destroys the parser instance.
431	Error	Raises an EParserError (311) exception with the given message.
431	ErrorFmt	Raises an EParserError (311) exception and formats the message.
431	ErrorStr	Raises an EParserError (311) exception with the given message.
431	HexToBinary	Writes hexadecimal data to a stream.
432	NextToken	Reads the next token and returns its type.
432	SourcePos	Returns the current position in the stream.
432	TokenComponentIdent	Returns the path of a subcomponent starting from the current token.
433	TokenFloat	Returns the current token as a float.
433	TokenInt	Returns the current token as an integer.
433	TokenString	Returns the current token as a string.
434	TokenSymbolIs	Returns `True` if the token equals the given symbol.
434	TokenWideString	Returns the current token as a widestring.

4.63.3 Property overview

Page	Properties	Access	Description
434	FloatType	r	The type of a float token.
435	SourceLine	r	Current source line number.
435	Token	r	The type of the current token.

4.63.4 TParser.Create

Synopsis: Creates a new parser instance.

Declaration: `constructor Create(Stream: TStream)`

Visibility: public

Description: `Create` creates a new `TParser` instance, using `Stream` as the stream to read data from, and reads the first token from the stream.

Errors: If an error occurs while parsing the first token, an EParserError (311) exception is raised.

See also: TParser.NextToken (432), TParser.Token (435)

4.63.5 TParser.Destroy

Synopsis: Destroys the parser instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Destroys the parser instance.

Errors: None.

4.63.6 TParser.CheckToken

Synopsis: Checks whether the token if of the given type.

Declaration: `procedure CheckToken(T: Char)`

Visibility: public

Description: Checks whether the token if of the given type.

Errors: If current token isn't of type `T`, an EParserError (311) exception is raised.

See also: TParser.Token (435)

4.63.7 TParser.CheckTokenSymbol

Synopsis: Checks whether the token equals the given symbol.

Declaration: `procedure CheckTokenSymbol(const S: string)`

Visibility: public

Description: `CheckTokenSymbol` performs a case-insensitive comparison of current token value with `S`.
Current token must be of type toSymbol (274), otherwise an EParserError (311) exception is raised.

Errors: If the comparison fails, or current token isn't a symbol, an EParserError (311) exception is raised.

See also: TParser.TokenSymbolIs (434), toSymbol (274)

4.63.8 TParser.Error

Synopsis: Raises an EParserError (311) exception with the given message.

Declaration: `procedure Error(const Ident: string)`

Visibility: public

Description: Raises an EParserError (311) exception with the given message.

4.63.9 TParser.ErrorFmt

Synopsis: Raises an EParserError (311) exception and formats the message.

Declaration: `procedure ErrorFmt(const Ident: string; const Args: Array of const)`

Visibility: public

Description: Raises an EParserError (311) exception and formats the message.

4.63.10 TParser.ErrorStr

Synopsis: Raises an EParserError (311) exception with the given message.

Declaration: `procedure ErrorStr(const Message: string)`

Visibility: public

Description: Raises an EParserError (311) exception with the given message.

4.63.11 TParser.HexToBinary

Synopsis: Writes hexadecimal data to a stream.

Declaration: `procedure HexToBinary(Stream: TStream)`

Visibility: public

Description: `HexToBinary` reads a sequence of hexadecimal characters from the input stream and converts them to a sequence of bytes which is written to `Stream`. Each byte is represented by two contiguous hexadecimal characters.

Whitespace is allowed between hexadecimal characters if it doesn't appear between two characters that form the same byte.

`HexToBinary` stops when the first non-hexadecimal and non-whitespace character is found, or the end of the input stream is reached.

Remark This method begins reading after the current token: that is, current token, even if it's a valid hexadecimal value, isn't included.

Errors: If a single hexadecimal character is found, an EParserError (311) exception is raised.

4.63.12 TParser.NextToken

Synopsis: Reads the next token and returns its type.

Declaration: `function NextToken : Char`

Visibility: public

Description: `NextToken` parses the next token in the stream and returns its type. The type of the token can also be retrieved later reading Token (435) property.

If the end of the stream is reached, toEOF (274) is returned.

For details about token types, see TParser.Token (435)

Errors: If an error occurs while parsing the token, an EParserError (311) exception is raised.

See also: TParser.Token (435)

4.63.13 TParser.SourcePos

Synopsis: Returns the current position in the stream.

Declaration: `function SourcePos : LongInt`

Visibility: public

Description **Remark** This is not the character position relative to the current source line, but the byte offset from the beginning of the stream.

Errors: None.

See also: TParser.SourceLine (435)

4.63.14 TParser.TokenComponentIdent

Synopsis: Returns the path of a subcomponent starting from the current token.

Declaration: `function TokenComponentIdent : string`

Visibility: public

Description: If current token is toSymbol (274), `TokenComponentIdent` tries to find subcomponent names separated by a dot (`.`). The returned string is the longest subcomponent path found. If there are no subcomponents, current symbol is returned.

Remark After this method has been called, subsequent calls to TokenString (433) or TokenWideString (434) return the same value returned by `TokenComponentIdent`.

Example

If source stream contains `a.b.c` and `TParser` is positioned on the first token (`a`), this method returns `a.b.c`.

Errors: If Token (435) isn't toSymbol (274), or no valid symbol is found after a dot, an EParserError (311) exception is raised.

See also: TParser.NextToken (432), TParser.Token (435), TParser.TokenString (433), TParser.TokenWideString (434), toSymbol (274)

4.63.15 TParser.TokenFloat

Synopsis: Returns the current token as a float.

Declaration: `function TokenFloat : Extended`

Visibility: public

Description: If current token type is toFloat (274), this method returns the token value as a float.

To specify a negative number, no space must exist between unary minus and number.

Floating point numbers can be postfixed with a character that specifies the floating point type. See FloatType (434) for further information.

Remark In the input stream the decimal separator, if present, must be a dot (`.`).

Errors: If Token (435) isn't toFloat (274), an EParserError (311) exception is raised.

See also: TParser.FloatType (434), TParser.NextToken (432), TParser.Token (435), toFloat (274)

4.63.16 TParser.TokenInt

Synopsis: Returns the current token as an integer.

Declaration: `function TokenInt : Int64`

Visibility: public

Description: If current token type is toInteger (274), this method returns the token value as an integer.

In the input stream an integer can be an hexadecimal (prefixed by `'$'` character) or decimal number. Decimal numbers can be prefixed by an unary minus: if this is the case, no space must exist between minus and number.

Errors: If Token (435) isn't toInteger (274), an EConvertError (272) exception is raised.

See also: TParser.NextToken (432), TParser.Token (435), toInteger (274)

4.63.17 TParser.TokenString

Synopsis: Returns the current token as a string.

Declaration: `function TokenString : string`

Visibility: public

Description: If current token type is toString (274) or toWString (274), this method returns the contents of the string. That is, enclosing quotes are removed, embedded quotes are unescaped and control strings are converted to the appropriate sequence of characters.

If current token type isn't a string, a string containing the token representation in the input stream is returned, without any conversion: hexadecimal integers are returned with the leading `$`, and floating point suffixes like `s`, `c` or `d` are kept. For tokens whose type isn't a special type, return value of `TokenString` equals Token (435).

Remark If Token (435) is toWString (274), TokenWideString (434) should be used instead.

Errors: None.

See also: TParser.NextToken (432), TParser.TokenWideString (434), TParser.Token (435), toString (274), toWString (274)

4.63.18 TParser.TokenWideString

Synopsis: Returns the current token as a widestring.

Declaration: `function TokenWideString : WideString`

Visibility: public

Description: If current token type is toWString (274), this method returns the contents of the string. That is, enclosing quotes are removed, embedded quotes are unescaped and control strings are converted to the appropriate sequence of characters.

If current token isn't a widestring, `TokenWideString` behaviour is the same as TokenString (433).

Errors: None.

See also: TParser.NextToken (432), TokenString (433), TParser.Token (435), toWString (274)

4.63.19 TParser.TokenSymbolIs

Synopsis: Returns `True` if the token equals the given symbol.

Declaration: `function TokenSymbolIs(const S: string) : Boolean`

Visibility: public

Description: `TokenSymbolIs` performs a case-insensitive comparison of current token value with `S`.

If current token isn't of type toSymbol (274), or comparison fails, `False` is returned.

Errors: None.

See also: TParser.CheckTokenSymbol (430), TParser.Token (435)

4.63.20 TParser.FloatType

Synopsis: The type of a float token.

Declaration: `Property FloatType : Char`

Visibility: public

Access: Read

Description: Floating point numbers can be postfixed with a character specifying the type of floating point value. When specified, this property holds the character postfixed to the number.

It can be one of the following values:

Table 4.28:

`s` or `S`	Value is a single.
`c` or `C`	Value is a currency.
`d` or `D`	Value is a date.

If Token (435) isn't toFloat (274) or one of the above characters wasn't specified, `FloatType` is the null character (zero).

See also: TParser.NextToken (432), TParser.Token (435), TParser.TokenFloat (433), toFloat (274)

4.63.21 TParser.SourceLine

Synopsis: Current source line number.

Declaration: `Property SourceLine : Integer`

Visibility: public

Access: Read

Description: Current source line number.

See also: TParser.SourcePos (432)

4.63.22 TParser.Token

Synopsis: The type of the current token.

Declaration: `Property Token : Char`

Visibility: public

Access: Read

Description: This property holds the type of the current token. When `Token` isn't one of the special token types (whose value can be retrieved with specific methods) it is the character representing the current token.

Special token types:

Table 4.29:

toEOF (274)	Value returned by TParser.Token (435) when the end of the input stream was reached.
toSymbol (274)	Value returned by TParser.Token (435) when a symbol was found in the input stream.
toString (274)	Value returned by TParser.Token (435) when a string was found in the input stream.
toInteger (274)	Value returned by TParser.Token (435) when an integer was found in the input stream.
toFloat (274)	Value returned by TParser.Token (435) when a floating point value was found in the input stream.
toWString (274)	Value returned by TParser.Token (435) when a widestring was found in the input stream.

To advance to the next token, use NextToken (432) method.

See also: TParser.CheckToken (430), TParser.NextToken (432), TParser.TokenComponentIdent (432), TParser.TokenFloat (433), TParser.TokenInt (433), TParser.TokenString (433), TParser.TokenWideString (434)

4.64 TPersistent

4.64.1 Description

`TPersistent` is the basic class for the streaming system. Since it is compiled in the `{$M+}` state, the compiler generates RTTI (Run-Time Type Information) for it and all classes that descend from it. This information can be used to stream all properties of classes.

It also introduces functionality to assign the contents of 2 classes to each other.

`TPersistent` implements the IFPObserved (314) interface for the benefit of descendant classes, but does not call IFPObserved.FPONotifyObservers (315). Descendants such as TStrings (475) and TCollection (365) and TCollectionItem (373) do use it.

See also: TComponent (376), IFPObserved (314), TStrings (475), TCollection (365), TCollectionItem (373)

4.64.2 Interfaces overview

Page	Interfaces	Description
314	IFPObserved	Interface implemented by an object that can be observed.

4.64.3 Method overview

Page	Method	Description
436	Assign	Assign the contents of one class to another.
436	Destroy	Destroys the `TPersistent` instance.
437	FPOAttachObserver	Add an observer to the list of observers.
437	FPODetachObserver	Remove an observer from the list of observers.
437	FPONotifyObservers	Notify observers of changes.
437	GetNamePath	Returns a string that can be used to identify the class instance.

4.64.4 TPersistent.Destroy

Synopsis: Destroys the `TPersistent` instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` disposes of the persistent object. This method should never be called directly. Instead the `Free` method should be used.

4.64.5 TPersistent.Assign

Synopsis: Assign the contents of one class to another.

Declaration: `procedure Assign(Source: TPersistent);  Virtual`

Visibility: public

Description: `Assign` copies the contents of `Source` to `Self`, if the classes of the destination and source classes are compatible.

The `TPersistent` implementation of `Assign` does nothing but calling the AssignTo (435) method of source. This means that if the destination class does not know how to assign the contents of the source class, the source class instance is asked to assign itself to the destination class. This means that it is necessary to implement only one of the two methods so that two classes can be assigned to one another.

Remark In general, a statement of the form

```
Destination:=Source;
```

(where `Destination` and `Source` are classes) does not achieve the same as a statement of the form

```
Destination.Assign(Source);
```

After the former statement, both `Source` and `Destination` will point to the same object. The latter statement will copy the *contents* of the `Source` class to the `Destination` class.

See also: AssignTo (435)

4.64.6 TPersistent.FPOAttachObserver

Synopsis: Add an observer to the list of observers.

Declaration:
```
procedure FPOAttachObserver(AObserver: TObject)
```

Visibility: public

Description: `FPOAttachObserver` is part of the implementation of the IFPObserved (314) interface in `TPersistent`. It adds a new observer to the list of observers. Calling this multiple times will add the observed object multiple times to the list.

Errors: An `EObserver` exception may be raised if `AObject` does not implement the IFPObserver (315) interface.

See also: IFPObserver (315), IFPObserved.FPOAttachObserver (314), IFPObserved (314)

4.64.7 TPersistent.FPODetachObserver

Synopsis: Remove an observer from the list of observers.

Declaration:
```
procedure FPODetachObserver(AObserver: TObject)
```

Visibility: public

Description: `FPODetachObserver` is part of the implementation of the IFPObserved (314) interface in `TPersistent`. It removes the first found instance of the observer from the list of observers.

See also: IFPObserved (314), IFPObserved.FPODetachObserver (315), IFPObserver (315)

4.64.8 TPersistent.FPONotifyObservers

Synopsis: Notify observers of changes.

Declaration:
```
procedure FPONotifyObservers(ASender: TObject;
                             AOperation: TFPObservedOperation;
                             Data: Pointer)
```

Visibility: public

Description: `FPONotifyObservers` can be called to notify observers of changes in the object. This method simply passes on the parameters that it receives to all attached IFPObserver (315) interfaces.

`TPersistent` does not call `FPONotifyObservers`. It is implemented for the benefit of descendant classes.

See also: IFPObserved (314), IFPObserved.FPONotifyObservers (315), IFPObserver (315)

4.64.9 TPersistent.GetNamePath

Synopsis: Returns a string that can be used to identify the class instance.

Declaration:
```
function GetNamePath : string;  Virtual
```

Visibility: public

Description: `GetNamePath` returns a string that can be used to identify the class instance. This can be used to display a name for this instance in a Object designer.

`GetNamePath` constructs a name by recursively prepending the `Classname` of the `Owner` instance to the `Classname` of this instance, separated by a dot.

See also: TPersistent.GetOwner (435)

4.65 TProxyStream

4.65.1 Description

`TProxyStream` is a proxy class for the #rtl.types.IStream (2012) interface. It implements all stream methods by relaying them to the `IStream` interface.

See also: #rtl.types.IStream (2012), TStreamAdapter (465)

4.65.2 Method overview

Page	Method	Description
439	Check	Check errors.
438	Create	Create a new instance of the `TProxyStream` class.
438	Read	
438	Seek	
438	Write	

4.65.3 TProxyStream.Create

Synopsis: Create a new instance of the `TProxyStream` class.

Declaration:
```
constructor Create(const Stream: IStream)
```

Visibility: public

Description: `Create` initializes a new instance of the `TProxyStream` class. It saves `var` stream for use in the other methods.

See also: #rtl.types.IStream (2012)

4.65.4 TProxyStream.Read

Declaration:
```
function Read(var Buffer; Count: LongInt) : LongInt;  Override
```

Visibility: public

4.65.5 TProxyStream.Write

Declaration:
```
function Write(const Buffer; Count: LongInt) : LongInt;  Override
```

Visibility: public

4.65.6 TProxyStream.Seek

Declaration:
```
function Seek(const Offset: Int64; Origin: TSeekOrigin) : Int64
                ;  Override
```

Visibility: public

4.65.7 TProxyStream.Check

Synopsis: Check errors.

Declaration: `procedure Check(err: Integer);  Virtual`

Visibility: public

Description: `Check` will check the result of the `IStream` interface. This method must be overridden by descendant classes to return interface-specific errors.

See also: #rtl.types.IStream (2012)

4.66 TRawByteStringStream

4.66.1 Description

`TRawByteStringStream` is a TBytesStream (364) descendent which introduces some auxiliary methods that help with inserting or extracting single-byte strings from the data: DataString (439)ReadString (440) and WriteString (440).

In difference with TStringStream (502), the data has no codepage information associated with it, so the data is always considered to use the default codepage.

See also: TStringStream (502), DataString (439), ReadString (440), WriteString (440)

4.66.2 Method overview

Page	Method	Description
439	Create	Create a new instance of `TRawByteStringStream`.
439	DataString	Return the stream contents as a single-byte string.
440	ReadString	Read a string from the stream.
440	WriteString	Write a string to the stream.

4.66.3 TRawByteStringStream.Create

Synopsis: Create a new instance of `TRawByteStringStream`.

Declaration: `constructor Create(const aData: RawByteString);  Overload`

Visibility: public

Description: `Create` creates a new instance of `TRawByteStringStream` and initalizes the Memory (389) with the characters in `aData`.

See also: Memory (389), DataString (439)

4.66.4 TRawByteStringStream.DataString

Synopsis: Return the stream contents as a single-byte string.

Declaration: `function DataString : RawByteString`

Visibility: public

Description: `DataString` returns the data in Memory (389) as a single-byte string without codepage information associated with it. The operation does not change the current position of the stream.

See also: Memory (389), ReadString (440), WriteString (440)

4.66.5 TRawByteStringStream.ReadString

Synopsis: Read a string from the stream.

Declaration: `function ReadString(Count: LongInt) : RawByteString`

Visibility: public

Description: `ReadString` reads at most `Count` bytes from the stream and returns the data as a single-byte string. The operation forwards the current position of the stream with at most `Count` bytes.

See also: DataString (439), WriteString (440)

4.66.6 TRawByteStringStream.WriteString

Synopsis: Write a string to the stream.

Declaration: `procedure WriteString(const AString: RawByteString)`

Visibility: public

Description: `WriteString` writes the bytes in `aString` to the stream. The operation forwards the current position of the stream with the length of the stream.

See also: DataString (439), WriteString (440)

4.67 TReader

4.67.1 Description

The `TReader` class is a reader class that implements generic component streaming capabilities, independent of the format of the data in the stream. It uses a driver class TAbstractObjectReader (325) to do the actual reading of data. The interface of the `TReader` class should be identical to the interface in Delphi.

Note that the `TReader` design is such that it can read a single component from a stream. It will read all children of this component, but it is not designed to read multiple components in succession from one stream.

It should never be necessary to create an instance of this class directly. Instead, the TStream.ReadComponent (459) call should be used.

See also: TFiler (393), TWriter (524), TAbstractObjectReader (325)

4.67.2 Method overview

Page	Method	Description
443	BeginReferences	Initializes the component referencing mechanism.
443	CheckValue	Raises an exception if the next value in the stream is not of type `Value`.
449	CopyValue	Copy a value to a writer.
442	Create	Creates a new reader class.
443	DefineBinaryProperty	Reads a user-defined binary property from the stream.
443	DefineProperty	Reads a user-defined property from the stream.
442	Destroy	Destroys a reader class.
443	EndOfList	Returns true if the stream contains an end-of-list marker.
444	EndReferences	Finalizes the component referencing mechanism.
444	FixupReferences	Tries to resolve all unresolved component references.
442	FlushBuffer	Flush the buffer.
444	NextValue	Returns the type of the next value.
444	Read	Read raw data from stream.
444	ReadBoolean	Reads a boolean from the stream.
445	ReadChar	Reads a character from the stream.
445	ReadCollection	Reads a collection from the stream.
445	ReadComponent	Starts reading a component from the stream.
446	ReadComponents	Starts reading child components from the stream.
446	ReadCurrency	Read a currency value from the stream.
446	ReadDate	Reads a date from the stream.
446	ReadFloat	Reads a float from the stream.
447	ReadIdent	Reads an identifier from the stream.
447	ReadInt64	Reads a 64-bit integer from the stream.
447	ReadInteger	Reads an integer from the stream.
447	ReadListBegin	Checks for the beginning of a list.
448	ReadListEnd	Checks for the end of a list.
448	ReadRootComponent	Starts reading a root component.
447	ReadSet	Read a set value from the stream.
448	ReadSignature	Read stream signature from the stream.
446	ReadSingle	Reads a single-type real from the stream.
448	ReadString	Reads a string from the stream.
445	ReadUnicodeChar	Read Unicode character.
449	ReadUnicodeString	Read a UnicodeString value from the stream.
449	ReadValue	Reads the next value type from the stream.
448	ReadVariant	Read a variant from the stream.
445	ReadWideChar	Read widechar from the stream.
449	ReadWideString	Read a WideString value from the stream.

4.67.3 Property overview

Page	Properties	Access	Description
449	Driver	r	The driver in use for streaming the data.
451	OnAncestorNotFound	rw	Handler called when the ancestor component cannot be found.
452	OnCreateComponent	rw	Handler called when a component needs to be created.
450	OnError	rw	Handler called when an error occurs.
452	OnFindComponentClass	rw	Handler called when a component class reference needs to be found.
451	OnFindMethod	rw	Handler to find or change a method address.
450	OnPropertyNotFound	rw	Handler for treating missing properties.
452	OnReadStringProperty	rw	Handler for translating strings when read from the stream.
451	OnReferenceName	rw	Handler called when another component is referenced.
451	OnSetMethodProperty	rw	Handler for setting method properties.
451	OnSetName	rw	Handler called when setting a component name.
450	Owner	rw	Owner of the component being read.
450	Parent	rw	Parent of the component being read.

4.67.4 TReader.Create

Synopsis: Creates a new reader class.

Declaration: `constructor Create(Stream: TStream; BufSize: Integer)`

Visibility: public

Description: Creates a new reader class.

4.67.5 TReader.Destroy

Synopsis: Destroys a reader class.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Destroys a reader class.

4.67.6 TReader.FlushBuffer

Synopsis: Flush the buffer.

Declaration: `procedure FlushBuffer;  Override`

Visibility: public

Description: `FlushBuffer` flushes the buffer. It is provided for Delphi compatibility, and is not used in FPC.

See also: TFiler.FlushBuffer (394), TAbstractObjectReader.FlushBuffer (326)

4.67.7 TReader.BeginReferences

Synopsis: Initializes the component referencing mechanism.

Declaration:
```
procedure BeginReferences
```

Visibility: public

Description: When streaming components, the streaming mechanism keeps a list of existing components that can be referenced to. This method initializes up that system.

4.67.8 TReader.CheckValue

Synopsis: Raises an exception if the next value in the stream is not of type `Value`.

Declaration:
```
procedure CheckValue(Value: TValueType)
```

Visibility: public

Description: Raises an exception if the next value in the stream is not of type `Value`.

4.67.9 TReader.DefineProperty

Synopsis: Reads a user-defined property from the stream.

Declaration:
```
procedure DefineProperty(const Name: string; AReadData: TReaderProc;
                         WriteData: TWriterProc; HasData: Boolean)
                         ;  Override
```

Visibility: public

Description: Reads a user-defined property from the stream.

4.67.10 TReader.DefineBinaryProperty

Synopsis: Reads a user-defined binary property from the stream.

Declaration:
```
procedure DefineBinaryProperty(const Name: string;
                               AReadData: TStreamProc;
                               WriteData: TStreamProc; HasData: Boolean)
                               ;  Override
```

Visibility: public

Description: Reads a user-defined binary property from the stream.

4.67.11 TReader.EndOfList

Synopsis: Returns true if the stream contains an end-of-list marker.

Declaration:
```
function EndOfList : Boolean
```

Visibility: public

Description: Returns true if the stream contains an end-of-list marker.

4.67.12 TReader.EndReferences

Synopsis: Finalizes the component referencing mechanism.

Declaration: `procedure EndReferences`

Visibility: public

Description: When streaming components, the streaming mechanism keeps a list of existing components that can be referenced to. This method cleans up that system.

4.67.13 TReader.FixupReferences

Synopsis: Tries to resolve all unresolved component references.

Declaration: `procedure FixupReferences`

Visibility: public

Description: Tries to resolve all unresolved component references.

4.67.14 TReader.NextValue

Synopsis: Returns the type of the next value.

Declaration: `function NextValue : TValueType`

Visibility: public

Description: Returns the type of the next value.

4.67.15 TReader.Read

Synopsis: Read raw data from stream.

Declaration: `procedure Read(var Buf; Count: LongInt);  Virtual`

Visibility: public

Description: `Read` is introduced for Delphi compatibility to read raw data from the component stream. This should not be used in new production code as it will totally mess up the streaming.

See also: TAbstractObjectReader.Read (328), TBinaryObjectReader.Read (348)

4.67.16 TReader.ReadBoolean

Synopsis: Reads a boolean from the stream.

Declaration: `function ReadBoolean : Boolean`

Visibility: public

Description: Reads a boolean from the stream.

4.67.17 TReader.ReadChar

Synopsis: Reads a character from the stream.

Declaration: `function ReadChar : Char`

Visibility: public

Description: Reads a character from the stream.

4.67.18 TReader.ReadWideChar

Synopsis: Read widechar from the stream.

Declaration: `function ReadWideChar : WideChar`

Visibility: public

Description: `TReader.ReadWideChar` reads a widechar from the stream. This actually reads a widestring and returns the first character.

See also: TReader.ReadWideString (449), TWriter.WriteWideChar (527)

4.67.19 TReader.ReadUnicodeChar

Synopsis: Read Unicode character.

Declaration: `function ReadUnicodeChar : UnicodeChar`

Visibility: public

Description: `ReadUnicodeChar` reads a single Unicode character from the stream. It does this by reading a `UnicodeString` string from the stream and returning the first character.

Errors: If the string has a length different from 1, an `EReadError` exception will occur.

See also: TReader.ReadUnicodeString (449)

4.67.20 TReader.ReadCollection

Synopsis: Reads a collection from the stream.

Declaration: `procedure ReadCollection(Collection: TCollection)`

Visibility: public

Description: Reads a collection from the stream.

4.67.21 TReader.ReadComponent

Synopsis: Starts reading a component from the stream.

Declaration: `function ReadComponent(Component: TComponent) : TComponent`

Visibility: public

Description: Starts reading a component from the stream.

4.67.22 TReader.ReadComponents

Synopsis: Starts reading child components from the stream.

Declaration:
```
procedure ReadComponents(AOwner: TComponent; AParent: TComponent;
                         Proc: TReadComponentsProc)
```

Visibility: public

Description: Starts reading child components from the stream.

4.67.23 TReader.ReadFloat

Synopsis: Reads a float from the stream.

Declaration:
```
function ReadFloat : Extended
```

Visibility: public

Description: Reads a float from the stream.

4.67.24 TReader.ReadSingle

Synopsis: Reads a single-type real from the stream.

Declaration:
```
function ReadSingle : Single
```

Visibility: public

Description: Reads a single-type real from the stream.

4.67.25 TReader.ReadDate

Synopsis: Reads a date from the stream.

Declaration:
```
function ReadDate : TDateTime
```

Visibility: public

Description: Reads a date from the stream.

4.67.26 TReader.ReadCurrency

Synopsis: Read a currency value from the stream.

Declaration:
```
function ReadCurrency : Currency
```

Visibility: public

Description: `ReadCurrency` reads a currency typed value from the stream and returns the result. This method does nothing except call the driver method of the driver being used.

See also: TWriter.WriteCurrency (528)

4.67.27 TReader.ReadIdent

Synopsis: Reads an identifier from the stream.

Declaration: `function ReadIdent : string`

Visibility: public

Description: Reads an identifier from the stream.

4.67.28 TReader.ReadInteger

Synopsis: Reads an integer from the stream.

Declaration: `function ReadInteger : LongInt`

Visibility: public

Description: Reads an integer from the stream.

4.67.29 TReader.ReadInt64

Synopsis: Reads a 64-bit integer from the stream.

Declaration: `function ReadInt64 : Int64`

Visibility: public

Description: Reads a 64-bit integer from the stream.

4.67.30 TReader.ReadSet

Synopsis: Read a set value from the stream.

Declaration: `function ReadSet(EnumType: Pointer) : Integer`

Visibility: public

Description: `ReadSet` reads a set of elements with type `EnumType` and returns them as an integer where each element is encoded in a bit of the integer. Thus, at most an enumerated type with 32 elements can be read with this function.

Errors: No checking is performed on the validity of `EnumType`. It is assumed to be a valid `PTypeInfo` pointer.

See also: TWriter.WriteSet (529)

4.67.31 TReader.ReadListBegin

Synopsis: Checks for the beginning of a list.

Declaration: `procedure ReadListBegin`

Visibility: public

Description: Checks for the beginning of a list.

4.67.32 TReader.ReadListEnd

Synopsis: Checks for the end of a list.

Declaration: `procedure ReadListEnd`

Visibility: public

Description: Checks for the end of a list.

4.67.33 TReader.ReadRootComponent

Synopsis: Starts reading a root component.

Declaration: `function ReadRootComponent(ARoot: TComponent) : TComponent`

Visibility: public

Description: Starts reading a root component.

4.67.34 TReader.ReadVariant

Synopsis: Read a variant from the stream.

Declaration: `function ReadVariant : Variant`

Visibility: public

Description: `ReadVariant` reads the next value from the stream and returns it as a variant. No variant array can be read from the stream, only single values.

Errors: If no variant manager is installed, the function will raise an `EReadError` exception. If the next value is not a simple value, again an `EReadError` exception is raised. exception is

See also: TBinaryObjectWriter.WriteVariant (358)

4.67.35 TReader.ReadSignature

Synopsis: Read stream signature from the stream.

Declaration: `procedure ReadSignature`

Visibility: public

Description: `ReadSignature` is called when starting to read a root component from a stream. Some streams contain a signature (header) to detect whether the stream contains correct data.

Errors: If the stream does not start with the correct signature, an EReadError (311) exception will be raised.

See also: TAbstractObjectReader.ReadSignature (332), TAbstractObjectWriter.WriteSignature (334)

4.67.36 TReader.ReadString

Synopsis: Reads a string from the stream.

Declaration: `function ReadString : string`

Visibility: public

Description: Reads a string from the stream.

4.67.37 TReader.ReadWideString

Synopsis: Read a WideString value from the stream.

Declaration: `function ReadWideString : WideString`

Visibility: public

Description: `ReadWidestring` reads a widestring typed value from the stream and returns the result. This method does nothing except call the driver method of the driver being used.

See also: TWriter.WriteWideString (530)

4.67.38 TReader.ReadUnicodeString

Synopsis: Read a UnicodeString value from the stream.

Declaration: `function ReadUnicodeString : UnicodeString`

Visibility: public

Description: `ReadUnicodeString` reads a `UnicodeString` string from the stream. The stream can contain a string from any type, it will be converted to UniCodeString.

See also: TAbstractObjectReader.ReadUnicodeString (333), TWriter.WriteUnicodeString (530)

4.67.39 TReader.ReadValue

Synopsis: Reads the next value type from the stream.

Declaration: `function ReadValue : TValueType`

Visibility: public

Description: Reads the next value type from the stream.

4.67.40 TReader.CopyValue

Synopsis: Copy a value to a writer.

Declaration: `procedure CopyValue(Writer: TWriter)`

Visibility: public

Description: `CopyValue` reads the next value from the reader stream, and writes it to the passed `Writer`.

4.67.41 TReader.Driver

Synopsis: The driver in use for streaming the data.

Declaration: `Property Driver : TAbstractObjectReader`

Visibility: public

Access: Read

Description: The driver in use for streaming the data.

4.67.42 TReader.Owner

Synopsis: Owner of the component being read.

Declaration: `Property Owner : TComponent`

Visibility: public

Access: Read,Write

Description: Owner of the component being read.

4.67.43 TReader.Parent

Synopsis: Parent of the component being read.

Declaration: `Property Parent : TComponent`

Visibility: public

Access: Read,Write

Description: Parent of the component being read.

4.67.44 TReader.OnError

Synopsis: Handler called when an error occurs.

Declaration: `Property OnError : TReaderError`

Visibility: public

Access: Read,Write

Description: Handler called when an error occurs.

4.67.45 TReader.OnPropertyNotFound

Synopsis: Handler for treating missing properties.

Declaration: `Property OnPropertyNotFound : TPropertyNotFoundEvent`

Visibility: public

Access: Read,Write

Description: `OnPropertyNotFound` can be used to take appropriate action when a property is read from a stream and no such property is found in the RTTI information of the Instance that is being read from the stream. It can be set at runtime, or at designtime by an IDE.

For more information about the meaning of the various arguments to the event handler, see TPropertyNotFoundEvent (285).

See also: TPropertyNotFoundEvent (285), TReader.OnSetMethodProperty (451), TReader.OnReadStringProperty (452)

4.67.46 TReader.OnFindMethod

Synopsis: Handler to find or change a method address.

Declaration: `Property OnFindMethod : TFindMethodEvent`

Visibility: public

Access: Read,Write

Description: Handler to find or change a method address.

4.67.47 TReader.OnSetMethodProperty

Synopsis: Handler for setting method properties.

Declaration: `Property OnSetMethodProperty : TSetMethodPropertyEvent`

Visibility: public

Access: Read,Write

Description: `OnSetMethodProperty` can be set to handle the setting of method properties. This handler can be used by an IDE to prevent methods from actually being assigned when an object is being streamed in the designer.

See also: TReader.OnReadStringProperty (452), TReader.OnPropertyNotFound (450)

4.67.48 TReader.OnSetName

Synopsis: Handler called when setting a component name.

Declaration: `Property OnSetName : TSetNameEvent`

Visibility: public

Access: Read,Write

Description: Handler called when setting a component name.

4.67.49 TReader.OnReferenceName

Synopsis: Handler called when another component is referenced.

Declaration: `Property OnReferenceName : TReferenceNameEvent`

Visibility: public

Access: Read,Write

Description: Handler called when another component is referenced.

4.67.50 TReader.OnAncestorNotFound

Synopsis: Handler called when the ancestor component cannot be found.

Declaration: `Property OnAncestorNotFound : TAncestorNotFoundEvent`

Visibility: public

Access: Read,Write

Description: Handler called when the ancestor component cannot be found.

4.67.51 TReader.OnCreateComponent

Synopsis: Handler called when a component needs to be created.

Declaration: `Property OnCreateComponent : TCreateComponentEvent`

Visibility: public

Access: Read,Write

Description: Handler called when a component needs to be created.

4.67.52 TReader.OnFindComponentClass

Synopsis: Handler called when a component class reference needs to be found.

Declaration: `Property OnFindComponentClass : TFindComponentClassEvent`

Visibility: public

Access: Read,Write

Description: Handler called when a component class reference needs to be found.

4.67.53 TReader.OnReadStringProperty

Synopsis: Handler for translating strings when read from the stream.

Declaration: `Property OnReadStringProperty : TReadWriteStringPropertyEvent`

Visibility: public

Access: Read,Write

Description: `OnReadStringProperty` is called whenever a string property is read from the stream. It can be used e.g. by a translation mechanism to translate the strings on the fly, when a form is loaded. See TReadWriteStringPropertyEvent (285) for a description of the various parameters.

See also: TReader.OnPropertyNotFound (450), TReader.OnSetMethodProperty (451), TReadWriteStringPropertyEvent (285)

4.68 TRecall

4.68.1 Description

`TRecall` is a helper class used to copy published properties of a class (the reference object) in another class (the storage object). The reference object and storage object must be assignable to each other.

The `TRecall` can be used to store the state of a persistent class, and restore it at a later time.

When a `TRecall` object is created, it gets passed a reference instance and a storage instance. It immediately stores the properties of the reference object in the storage object.

The Store (453) method can be called throughout the lifetime of the reference object to update the stored properties.

When the `TRecall` instance is destroyed then the properties are copied from the storage object to the reference object. The storage object is freed automatically.

If the properties should not be copied back from the storage to the reference object, the Forget (454) can be called.

See also: TRecall.Create (453), TRecall.Destroy (453), TRecall.Forget (454), TRecall.Store (453), TPersistent.Assign (436)

4.68.2 Method overview

Page	Method	Description
453	Create	Creates a new instance of `TRecall`.
453	Destroy	Copies the stored properties to the reference object and destroys the `TRecall` instance.
454	Forget	Clear the reference property.
453	Store	Assigns the reference instance to the storage instance.

4.68.3 Property overview

Page	Properties	Access	Description
454	Reference	r	The reference object.

4.68.4 TRecall.Create

Synopsis: Creates a new instance of `TRecall`.

Declaration: `constructor Create(AStorage: TPersistent; AReference: TPersistent)`

Visibility: public

Description: `Create` creates a new instance of `TRecall` and initializes the Reference and Storage instances. It calls Store (453) to assign the reference object properties to the storage instance.

See also: TRecall.Store (453), TRecall.Destroy (453)

4.68.5 TRecall.Destroy

Synopsis: Copies the stored properties to the reference object and destroys the `TRecall` instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` assigns the storage instance to the reference instance, if the latter is still valid. After this, it frees the storage and calls the inherited destroy.

Errors: `Destroy` does not check whether the reference (454) instance is still valid. If the reference pointer was invalidated, call TRecall.Forget (454) to clear the reference instance.

See also: TRecall.Store (453), TRecall.Forget (454)

4.68.6 TRecall.Store

Synopsis: Assigns the reference instance to the storage instance.

Declaration: `procedure Store`

Visibility: public

Description: `Store` assigns the reference instance to the storage instance. This will only work if the two classes can be assigned to each other.

This method can be used to refresh the storage.

Errors: `Store` does not check whether the reference (454) instance is still valid. If the reference pointer was invalidated, call TRecall.Forget (454) to clear the reference instance.

4.68.7 TRecall.Forget

Synopsis: Clear the reference property.

Declaration: `procedure Forget`

Visibility: public

Description: `Forget` sets the Reference (454) property to `Nil`. When the `TRecall` instance is destroyed, the reference instance will not be restored.

Note that after a call to `Forget`, a call to Store (453) has no effect.

Errors: None.

See also: TRecall.Reference (454), TRecall.Store (453), TRecall.Destroy (453)

4.68.8 TRecall.Reference

Synopsis: The reference object.

Declaration: `Property Reference : TPersistent`

Visibility: public

Access: Read

Description: `Reference` is the instance of the reference object. Do not free the reference directly. Call Forget (454) to clear the reference and then free the reference object.

See also: TRecall.Forget (454)

4.69 TResourceStream

4.69.1 Description

Stream that reads its data from a resource object.

4.69.2 Method overview

Page	Method	Description
455	Create	Creates a new instance of a resource stream.
455	CreateFromID	Creates a new instance of a resource stream with a resource.
455	Destroy	Destroys the instance of the resource stream.

4.69.3 TResourceStream.Create

Synopsis: Creates a new instance of a resource stream.

Declaration:
```
constructor Create(Instance: TFPResourceHMODULE; const ResName: string;
                   ResType: PChar)
```

Visibility: public

Description: Creates a new instance of a resource stream.

4.69.4 TResourceStream.CreateFromID

Synopsis: Creates a new instance of a resource stream with a resource.

Declaration:
```
constructor CreateFromID(Instance: TFPResourceHMODULE; ResID: Integer;
                         ResType: PChar)
```

Visibility: public

Description: The resource is loaded from the loaded module (identified by the handle `Instance`), identifier `ResID` and type `ResType`.

4.69.5 TResourceStream.Destroy

Synopsis: Destroys the instance of the resource stream.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: Destroys the instance of the resource stream.

4.70 TStream

4.70.1 Description

`TStream` is the base class for all streaming classes. It defines methods for reading (456), writing (457) from and to streams, as well as functions to determine the size of the stream as well as the current position of the stream.

Descendant classes such as TMemoryStream (425) or TFileStream (395) then override these methods to write streams to memory or file.

See also: TMemoryStream (425), TFileStream (395), TStringStream (502)

4.70.2 Method overview

Page	Method	Description
459	CopyFrom	Copy data from one stream to another.
461	FixupResourceHeader	Not implemented in FPC.
456	Read	Reads data from the stream to a buffer and returns the number of bytes read.
463	ReadAnsiString	Read an ansistring from the stream and return its value.
458	ReadBuffer	Reads data from the stream to a buffer.
461	ReadByte	Read a byte from the stream and return its value.
459	ReadComponent	Reads component data from a stream.
459	ReadComponentRes	Reads component data and resource header from a stream.
462	ReadDWord	Read a DWord from the stream and return its value.
462	ReadQWord	Read a QWord value from the stream and return its value.
461	ReadResHeader	Read a resource header from the stream.
462	ReadWord	Read a word from the stream and return its value.
457	Seek	Sets the current position in the stream.
457	Write	Writes data from a buffer to the stream and returns the number of bytes written.
464	WriteAnsiString	Write an ansistring to the stream.
458	WriteBuffer	Writes data from a buffer to the stream.
463	WriteByte	Write a byte to the stream.
460	WriteComponent	Write component data to the stream.
460	WriteComponentRes	Write resource header and component data to a stream.
460	WriteDescendent	Write descendent of a component.
460	WriteDescendentRes	Write descendent of a component as resource.
463	WriteDWord	Write a DWord to the stream.
464	WriteQWord	Write a QWord value to the stream.
461	WriteResourceHeader	Write resource header to the stream.
463	WriteWord	Write a word to the stream.

4.70.3 Property overview

Page	Properties	Access	Description
464	Position	rw	The current position in the stream.
465	Size	rw	The current size of the stream.

4.70.4 TStream.Read

Synopsis: Reads data from the stream to a buffer and returns the number of bytes read.

Declaration: `function Read(var Buffer; Count: LongInt) : LongInt; Virtual; Overload`

Visibility: public

Description: `Read` attempts to read `Count` from the stream to `Buffer` and returns the number of bytes actually read.

This method should be used when the number of bytes is not determined. If a specific number of bytes is expected, use TSTream.ReadBuffer (458) instead.

As implemented in `TStream`, `Read` does nothing but raises an EStreamError (312) exception to indicate that reading is not supported. Descendant classes that allow reading must override this method to do the actual reading.

Descendant classes should (if they don't explicitly raise an exception) return a positive value (>=0), where zero indicates an error.

Errors: In case a descendant class does not allow reading from the stream, an exception is raised.

See also: TStream.Write (457), TStream.ReadBuffer (458)

4.70.5 TStream.Write

Synopsis: Writes data from a buffer to the stream and returns the number of bytes written.

Declaration:
```
function Write(const Buffer; Count: LongInt) : LongInt;  Virtual
                ;  Overload
```

Visibility: public

Description: `Write` attempts to write `Count` bytes from `Buffer` to the stream. It returns the actual number of bytes written to the stream.

This method should be used when the number of bytes that should be written is not determined. If a specific number of bytes should be written, use TSTream.WriteBuffer (458) instead.

As implemented in `TStream`, `Write` does nothing but raises EStreamError (312) exception to indicate that writing is not supported. Descendant classes that allow writing must override this method to do the actual writing.

Descendant classes should (if they don't explicitly raise an exception) return a positive value (>=0), where zero indicates an error.

Errors: In case a descendant class does not allow writing to the stream, an exception is raised.

See also: TStream.Read (456), TStream.WriteBuffer (458)

4.70.6 TStream.Seek

Synopsis: Sets the current position in the stream.

Declaration:
```
function Seek(Offset: LongInt; Origin: Word) : LongInt;  Virtual
              ;  Overload
function Seek(const Offset: Int64; Origin: TSeekOrigin) : Int64
              ;  Virtual;  Overload
```

Visibility: public

Description: `Seek` sets the position of the stream to `Offset` bytes from `Origin`. There is a 32-bit variant of this function and a 64-bit variant. The difference can be made by choosing the correct `Offset` parameter: the integer-typed parameter selects the 32-bit variant, the parameter of type TSeekOrigin (285) selects the 64-bit variant of the function.

Both variants of this function return the new (0-based) position in the stream.

The `Origin` parameter for the 32-bit version can have one of the following values:

Table 4.30:

Constant	Meaning
soFromBeginning	Set the position relative to the start of the stream.
soFromCurrent	Set the position relative to the current position in the stream.
soFromEnd	Set the position relative to the end of the stream.

These values are defined in the SysUtils (272) unit.

The `Origin` parameter for the 64-bit version has one of the following values:

Table 4.31:

Value	Meaning
soBeginning	Offset is interpreted relative to the start of the stream.
soCurrent	Offset is interpreted relative to the current position in the stream.
soEnd	Offset is interpreted relative to the end of the stream.

`Offset` should be negative when the origin is `SoFromEnd` (`soEnd`). It should be positive for `soFromBeginning` and can have both signs for `soFromCurrent`

This is an abstract method, which must be overridden by descendant classes. They may choose not to implement this method for all values of `Origin` and `Offset`.

Remark Internally, all calls are re-routed to the 64-bit version of the call. When creating a descendant of `TStream`, the 64-bit version of the call should be overridden.

Errors: An exception may be raised if this method is called with an invalid pair of `Offset`,`Origin` values. e.g. a negative `offset` for `soFromBeginning` (or `soBeginning`).

See also: TStream.Position (464)

4.70.7 TStream.ReadBuffer

Synopsis: Reads data from the stream to a buffer.

Declaration: `procedure ReadBuffer(var Buffer; Count: LongInt)`

Visibility: public

Description: `ReadBuffer` reads `Count` bytes of the stream into `Buffer`. If the stream does not contain `Count` bytes, then an exception is raised.

`ReadBuffer` should be used to read in a fixed number of bytes, such as when reading structures or the content of variables. If the number of bytes is not determined, use TStream.Read (456) instead. `ReadBuffer` uses `Read` internally to do the actual reading.

Errors: If the stream does not allow to read `Count` bytes, then an exception is raised.

See also: TStream.Read (456), TStream.WriteBuffer (458)

4.70.8 TStream.WriteBuffer

Synopsis: Writes data from a buffer to the stream.

Declaration: `procedure WriteBuffer(const Buffer; Count: LongInt)`

Visibility: public

Description: `WriteBuffer` writes `Count` bytes to the stream from `Buffer`. If the stream does not allow `Count` bytes to be written, then an exception is raised.

`WriteBuffer` should be used to write a fixed number of bytes, such as when writing structures or the content of variables. If the number of bytes is not determined, use TStream.Write (457) instead. `WriteBuffer` uses `Write` internally to do the actual writing.

Errors: If the stream does not allow to write `Count` bytes, then an exception is raised.

See also: TStream.Write (457), TStream.ReadBuffer (458)

4.70.9 TStream.CopyFrom

Synopsis: Copy data from one stream to another.

Declaration: `function CopyFrom(Source: TStream; Count: Int64) : Int64`

Visibility: public

Description: `CopyFrom` reads `Count` bytes from `Source` and writes them to the current stream. This updates the current position in the stream. After the action is completed, the number of bytes copied is returned. If `Count` is zero, then the whole contents of the `Source` stream is copied. It is positioned on the first byte of data, and `Size` bytes are copied. Note that this cannot be used with streams that do not allow seeking or do not allow determining the size of the stream.

This can be used to quickly copy data from one stream to another or to copy the whole contents of the stream.

See also: TStream.Read (456), TStream.Write (457)

4.70.10 TStream.ReadComponent

Synopsis: Reads component data from a stream.

Declaration: `function ReadComponent(Instance: TComponent) : TComponent`

Visibility: public

Description: `ReadComponent` reads a component state from the stream and transfers this state to `Instance`. If `Instance` is nil, then it is created first based on the type stored in the stream. `ReadComponent` returns the component as it is read from the stream.

`ReadComponent` simply creates a TReader (440) object and calls its ReadRootComponent (448) method.

Errors: If an error occurs during the reading of the component, an EFilerError (310) exception is raised.

See also: TStream.WriteComponent (460), TStream.ReadComponentRes (459), TReader.ReadRootComponent (448)

4.70.11 TStream.ReadComponentRes

Synopsis: Reads component data and resource header from a stream.

Declaration: `function ReadComponentRes(Instance: TComponent) : TComponent`

Visibility: public

Description: `ReadComponentRes` reads a resource header from the stream, and then calls ReadComponent (459) to read the component state from the stream into `Instance`.

This method is usually called by the global streaming method when instantiating forms and datamodules as created by an IDE. It should be used mainly on Windows, to store components in Windows resources.

Errors: If an error occurs during the reading of the component, an EFilerError (310) exception is raised.

See also: TStream.ReadComponent (459), TStream.WriteComponentRes (460)

4.70.12 TStream.WriteComponent

Synopsis: Write component data to the stream.

Declaration: `procedure WriteComponent(Instance: TComponent)`

Visibility: public

Description: `WriteComponent` writes the published properties of `Instance` to the stream, so they can later be read with TStream.ReadComponent (459). This method is intended to be used by an IDE, to preserve the state of a form or datamodule as designed in the IDE.

`WriteComponent` simply calls WriteDescendant (455) with `Nil` ancestor.

See also: TStream.ReadComponent (459), TStream.WriteComponentRes (460)

4.70.13 TStream.WriteComponentRes

Synopsis: Write resource header and component data to a stream.

Declaration: `procedure WriteComponentRes(const ResName: string; Instance: TComponent)`

Visibility: public

Description: `WriteComponentRes` writes a `ResName` resource header to the stream and then calls WriteComponent (460) to write the published properties of `Instance` to the stream.

This method is intended for use by an IDE that can use it to store forms or datamodules as designed in a Windows resource stream.

See also: TStream.WriteComponent (460), TStream.ReadComponentRes (459)

4.70.14 TStream.WriteDescendent

Synopsis: Write descendent of a component.

Declaration: `procedure WriteDescendent(Instance: TComponent; Ancestor: TComponent)`

Visibility: public

Description: `WriteDescendent` will create a TWriter (524) writer class and write `Instance` as a descendent of `Ancestor` using the writer. This is used to create diff streams: only the properties where `Instance` differs from `Ancestor` are written to the stream.

See also: TWriter.WriteDescendent (528)

4.70.15 TStream.WriteDescendentRes

Synopsis: Write descendent of a component as resource.

Declaration:
```
procedure WriteDescendentRes(const ResName: string;
                             Instance: TComponent; Ancestor: TComponent)
```

Visibility: public

Description: `WriteDescendentRes` calls `WriteDescendent` as a resource stream.

See also: TWriter.WriteDescendent (528)

4.70.16 TStream.WriteResourceHeader

Synopsis: Write resource header to the stream.

Declaration:
```
procedure WriteResourceHeader(const ResName: string;
                              var FixupInfo: LongInt)
```

Visibility: public

Description: `WriteResourceHeader` writes a resource-file header for a resource called `ResName`. It returns in `FixupInfo` the argument that should be passed on to TStream.FixupResourceHeader (461).

`WriteResourceHeader` should not be used directly. It is called by the TStream.WriteComponentRes (460) and TStream.WriteDescendantRes (455) methods.

See also: TStream.FixupResourceHeader (461), TStream.WriteComponentRes (460), TStream.WriteDescendantRes (455)

4.70.17 TStream.FixupResourceHeader

Synopsis: Not implemented in FPC.

Declaration:
```
procedure FixupResourceHeader(FixupInfo: LongInt)
```

Visibility: public

Description: `FixupResourceHeader` is used to write the size of the resource after a component was written to stream. The size is determined from the current position, and it is written at position `FixupInfo`. After that the current position is restored.

`FixupResourceHeader` should never be called directly; it is handled by the streaming system.

See also: TStream.WriteResourceHeader (461), TStream.WriteComponentRes (460), TStream.WriteDescendantRes (455)

4.70.18 TStream.ReadResHeader

Synopsis: Read a resource header from the stream.

Declaration:
```
procedure ReadResHeader
```

Visibility: public

Description: `ReadResourceHeader` reads a resource file header from the stream. It positions the stream just beyond the header.

`ReadResourceHeader` should not be called directly, it is called by the streaming system when needed.

Errors: If the resource header is invalid an EInvalidImage (310) exception is raised.

See also: TStream.ReadComponentRes (459), EInvalidImage (310)

4.70.19 TStream.ReadByte

Synopsis: Read a byte from the stream and return its value.

Declaration:
```
function ReadByte : Byte
```

Visibility: public

Description: `ReadByte` reads one byte from the stream and returns its value.

Errors: If the byte cannot be read, a EStreamError (312) exception will be raised. This is a utility function which simply calls the Read (456) function.

See also: TStream.Read (456), TStream.WriteByte (463), TStream.ReadWord (462), TStream.ReadDWord (462), TStream.ReadAnsiString (463)

4.70.20 TStream.ReadWord

Synopsis: Read a word from the stream and return its value.

Declaration: `function ReadWord : Word`

Visibility: public

Description: `ReadWord` reads one Word (i.e. 2 bytes) from the stream and returns its value. This is a utility function which simply calls the Read (456) function.

Errors: If the word cannot be read, a EStreamError (312) exception will be raised.

See also: TStream.Read (456), TStream.WriteWord (463), TStream.ReadByte (461), TStream.ReadDWord (462), TStream.ReadAnsiString (463)

4.70.21 TStream.ReadDWord

Synopsis: Read a DWord from the stream and return its value.

Declaration: `function ReadDWord : Cardinal`

Visibility: public

Description: `ReadDWord` reads one DWord (i.e. 4 bytes) from the stream and returns its value. This is a utility function which simply calls the Read (456) function.

Errors: If the DWord cannot be read, a EStreamError (312) exception will be raised.

See also: TStream.Read (456), TStream.WriteDWord (463), TStream.ReadByte (461), TStream.ReadWord (462), TStream.ReadAnsiString (463)

4.70.22 TStream.ReadQWord

Synopsis: Read a QWord value from the stream and return its value.

Declaration: `function ReadQWord : QWord`

Visibility: public

Description: `ReadQWord` reads a `QWord` value (8 bytes) from the stream and returns it's value.

Errors: If not enough bytes are available on the stream, an EStreamError (312) exception will be raised.

See also: TStream.Read (456), TStream.WriteByte (463), TStream.ReadWord (462), TStream.ReadDWord (462), TStream.ReadAnsiString (463)

4.70.23 TStream.ReadAnsiString

Synopsis: Read an ansistring from the stream and return its value.

Declaration: `function ReadAnsiString : string`

Visibility: public

Description: `ReadAnsiString` reads an ansistring from the stream and returns its value. This is a utility function which simply calls the read function several times. The Ansistring should be stored as 4 bytes (a DWord) representing the length of the string, and then the string value itself. The WriteAnsiString (464) function writes an ansistring in such a format.

Errors: If the AnsiString cannot be read, a EStreamError (312) exception will be raised.

See also: TStream.Read (456), TStream.WriteAnsiString (464), TStream.ReadByte (461), TStream.ReadWord (462), TStream.ReadDWord (462)

4.70.24 TStream.WriteByte

Synopsis: Write a byte to the stream.

Declaration: `procedure WriteByte(b: Byte)`

Visibility: public

Description: `WriteByte` writes the byte `B` to the stream. This is a utility function which simply calls the Write (457) function. The byte can be read from the stream using the ReadByte (461) function.

Errors: If an error occurs when attempting to write, an EStreamError (312) exception will be raised.

See also: TStream.Write (457), TStream.ReadByte (461), TStream.WriteWord (463), TStream.WriteDWord (463), TStream.WriteAnsiString (464)

4.70.25 TStream.WriteWord

Synopsis: Write a word to the stream.

Declaration: `procedure WriteWord(w: Word)`

Visibility: public

Description: `WriteWord` writes the word `W` (i.e. 2 bytes) to the stream. This is a utility function which simply calls the Write (457) function. The word can be read from the stream using the ReadWord (462) function.

Errors: If an error occurs when attempting to write, an EStreamError (312) exception will be raised.

See also: TStream.Write (457), TStream.ReadWord (462), TStream.WriteByte (463), TStream.WriteDWord (463), TStream.WriteAnsiString (464)

4.70.26 TStream.WriteDWord

Synopsis: Write a DWord to the stream.

Declaration: `procedure WriteDWord(d: Cardinal)`

Visibility: public

Description: `WriteDWord` writes the DWord `D` (i.e. 4 bytes) to the stream. This is a utility function which simply calls the Write (457) function. The DWord can be read from the stream using the ReadDWord (462) function.

Errors: If an error occurs when attempting to write, an EStreamError (312) exception will be raised.

See also: TStream.Write (457), TStream.ReadDWord (462), TStream.WriteByte (463), TStream.WriteWord (463), TStream.WriteAnsiString (464)

4.70.27 TStream.WriteQWord

Synopsis: Write a QWord value to the stream.

Declaration: `procedure WriteQWord(q: QWord)`

Visibility: public

Description: `WriteQWord` writes the word `W` (i.e. 8 bytes) to the stream. This is a utility function which simply calls the Write (457) function. The word can be read from the stream using the ReadQWord (462) function.

Errors: If an error occurs when attempting to write, an EStreamError (312) exception will be raised.

See also: TStream.Write (457), TStream.ReadByte (461), TStream.WriteWord (463), TStream.WriteDWord (463), TStream.WriteAnsiString (464)

4.70.28 TStream.WriteAnsiString

Synopsis: Write an ansistring to the stream.

Declaration: `procedure WriteAnsiString(const S: string);  Virtual`

Visibility: public

Description: `WriteAnsiString` writes the AnsiString `S` (i.e. 4 bytes) to the stream. This is a utility function which simply calls the Write (457) function. The ansistring is written as a 4 byte length specifier, followed by the ansistring's content. The ansistring can be read from the stream using the ReadAnsiString (463) function.

Errors: If an error occurs when attempting to write, an EStreamError (312) exception will be raised.

See also: TStream.Write (457), TStream.ReadAnsiString (463), TStream.WriteByte (463), TStream.WriteWord (463), TStream.WriteDWord (463)

4.70.29 TStream.Position

Synopsis: The current position in the stream.

Declaration: `Property Position : Int64`

Visibility: public

Access: Read,Write

Description: `Position` can be read to determine the current position in the stream. It can be written to set the (absolute) position in the stream. The position is zero-based, so to set the position at the beginning of the stream, the position must be set to zero.

Remark Not all `TStream` descendants support setting the position in the stream, so this should be used with care.

Errors: Some descendants may raise an EStreamError (312) exception if they do not support setting the stream position.

See also: TStream.Size (465), TStream.Seek (457)

4.70.30 TStream.Size

Synopsis: The current size of the stream.

Declaration: `Property Size : Int64`

Visibility: public

Access: Read,Write

Description: `Size` can be read to determine the stream size or to set the stream size.

Remark Not all descendants of `TStream` support getting or setting the stream size; they may raise an exception if the `Size` property is read or set.

See also: TStream.Position (464), TStream.Seek (457)

4.71 TStreamAdapter

4.71.1 Description

Implements IStream for TStream (455) descendants.

4.71.2 Interfaces overview

Page	Interfaces	Description
2012	IStream	COM stream abstraction.

4.71.3 Method overview

Page	Method	Description
469	Clone	Clone the stream.
468	Commit	Commit data to the stream.
467	CopyTo	Copy data to destination stream.
466	Create	Create a new instance of `TStreamAdapter`.
466	Destroy	Free the `TStreamAdapter` instance.
468	LockRegion	Lock a region of the stream.
466	Read	Read from the stream.
468	Revert	Revert operations on the stream.
467	Seek	Set the stream position.
467	SetSize	Set the stream size.
469	Stat	Return statistical data about the stream.
468	UnlockRegion	Unlock a region of the stream.
466	Write	Write to the stream.

4.71.4 Property overview

Page	Properties	Access	Description
469	Stream	r	Stream on which adaptor works.
469	StreamOwnership	rw	Determines what happens with the stream when the adaptor is freed.

4.71.5 TStreamAdapter.Create

Synopsis: Create a new instance of `TStreamAdapter`.

Declaration:
```
constructor Create(Stream: TStream; Ownership: TStreamOwnership)
```

Visibility: public

Description: `Create` creates a new instance of `TStreamAdaptor`. It initializes TStreamAdapter.Stream (469) with `Stream` and initializes StreamOwnerShip (469) with `Ownership`.

`TStreamAdapter` is an abstract class: descendants must be created that implement the actual functionality.

See also: StreamOwnerShip (469), TStreamAdapter.Stream (469)

4.71.6 TStreamAdapter.Destroy

Synopsis: Free the `TStreamAdapter` instance.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: Explicitly free the `TStreamAdapter` instance. Normally, this is done automatically if a reference to the `IStream` interface is freed.

4.71.7 TStreamAdapter.Read

Synopsis: Read from the stream.

Declaration:
```
function Read(pv: Pointer; cb: DWORD; pcbRead: PDWord) : HRESULT
                  ;  Virtual
```

Visibility: public

Description: `Read` implements #rtl.types.ISequentialStream.Read (2011) by reading from the stream specified at creation.

Errors: This function must be overridden and will raise a runerror 217 when called directly.

See also: #rtl.types.ISequentialStream.Read (2011)

4.71.8 TStreamAdapter.Write

Synopsis: Write to the stream.

Declaration:
```
function Write(pv: Pointer; cb: DWORD; pcbWritten: PDWord) : HRESULT
                  ;  Virtual
```

Visibility: public

Description: `Write` implements #rtl.types.ISequentialStream.Write (2011) by writing to the stream specified at creation.

Errors: This function must be overridden and will raise a runerror 217 when called directly.

See also: #rtl.types.ISequentialStream.Write (2011)

4.71.9 TStreamAdapter.Seek

Synopsis: Set the stream position.

Declaration:
```
function Seek(dlibMove: Largeint; dwOrigin: DWORD;
                    out libNewPosition: LargeUint) : HRESULT;  Virtual
```

Visibility: public

Description: `Seek` implements #rtl.types.IStream.Seek (2012) by setting the position of the stream specified at creation.

Errors: This function must be overridden and will raise a runerror 217 when called directly.

See also: #rtl.types.IStream.Seek (2012)

4.71.10 TStreamAdapter.SetSize

Synopsis: Set the stream size.

Declaration:
```
function SetSize(libNewSize: LargeUint) : HRESULT;  Virtual
```

Visibility: public

Description: `SetSize` implements #rtl.types.IStream.Setsize (2012) by setting the size of the stream specified at creation.

Errors: This function must be overridden and will raise a runerror 217 when called directly.

See also: #rtl.types.IStream.Setsize (2012)

4.71.11 TStreamAdapter.CopyTo

Synopsis: Copy data to destination stream.

Declaration:
```
function CopyTo(stm: IStream; cb: LargeUint; out cbRead: LargeUint;
                    out cbWritten: LargeUint) : HRESULT;  Virtual
```

Visibility: public

Description: `CopyTo` implements #rtl.types.IStream.CopyTo (2012).

Errors: This function must be overridden and will raise a runerror 217 when called directly.

4.71.12 TStreamAdapter.Commit

Synopsis: Commit data to the stream.

Declaration:
```
function Commit(grfCommitFlags: DWORD) : HRESULT;  Virtual
```

Visibility: public

Description: `Commit` implements #rtl.types.IStream.Commit (2013).

Errors: This function must be overridden and will raise a runerror 217 when called directly.

See also: #rtl.types.IStream.Commit (2013)

4.71.13 TStreamAdapter.Revert

Synopsis: Revert operations on the stream.

Declaration:
```
function Revert : HRESULT;  Virtual
```

Visibility: public

Description: `Revert` implements #rtl.types.IStream.Revert (2013).

Errors: This function must be overridden and will raise a runerror 217 when called directly.

See also: #rtl.types.IStream.Revert (2013)

4.71.14 TStreamAdapter.LockRegion

Synopsis: Lock a region of the stream.

Declaration:
```
function LockRegion(libOffset: LargeUint; cb: LargeUint;
                    dwLockType: DWORD) : HRESULT;  Virtual
```

Visibility: public

Description: `LockRegion` implements #rtl.types.IStream.LockRegion (2013).

Errors: This function must be overridden and will raise a runerror 217 when called directly.

See also: #rtl.types.IStream.LockRegion (2013)

4.71.15 TStreamAdapter.UnlockRegion

Synopsis: Unlock a region of the stream.

Declaration:
```
function UnlockRegion(libOffset: LargeUint; cb: LargeUint;
                      dwLockType: DWORD) : HRESULT;  Virtual
```

Visibility: public

Description: `UnLockRegion` implements #rtl.types.IStream.UnLockRegion (2013).

Errors: This function must be overridden and will raise a runerror 217 when called directly.

See also: #rtl.types.IStream.UnLockRegion (2013)

4.71.16 TStreamAdapter.Stat

Synopsis: Return statistical data about the stream.

Declaration:
```
function Stat(out statstg: TStatStg; grfStatFlag: DWORD) : HRESULT
                    ;  Virtual
```

Visibility: public

Description: `Stat` implements #rtl.types.IStream.Stat (2014).

Errors: This function must be overridden and will raise a runerror 217 when called directly.

See also: #rtl.types.IStream.Stat (2014)

4.71.17 TStreamAdapter.Clone

Synopsis: Clone the stream.

Declaration:
```
function Clone(out stm: IStream) : HRESULT;  Virtual
```

Visibility: public

Description: `Clone` implements #rtl.types.IStream.Clone (2014).

Errors: This function must be overridden and will raise a runerror 217 when called directly.

See also: #rtl.types.IStream.Clone (2014)

4.71.18 TStreamAdapter.Stream

Synopsis: Stream on which adaptor works.

Declaration:
```
Property Stream : TStream
```

Visibility: public

Access: Read

Description: This is the stream on which the adaptor works. It was specified at creation.

4.71.19 TStreamAdapter.StreamOwnership

Synopsis: Determines what happens with the stream when the adaptor is freed.

Declaration:
```
Property StreamOwnership : TStreamOwnership
```

Visibility: public

Access: Read,Write

Description: `StreamOwnership` determines what happens when the adaptor

4.72 TStringList

4.72.1 Description

`TStringList` is a descendant class of TStrings (475) that implements all of the abstract methods introduced there. It also introduces some additional methods:

- Sort the list, or keep the list sorted at all times
- Special handling of duplicates in sorted lists
- Notification of changes in the list

See also: TStrings (475), TStringList.Duplicates (473), TStringList.Sorted (474)

4.72.2 Method overview

Page	Method	Description
471	Add	Implements the TStrings.Add (480) function.
471	Clear	Implements the TStrings.Clear (483) function.
470	Create	
473	CustomSort	Sort the stringlist using a custom sort algorithm.
471	Delete	Implements the TStrings.Delete (483) function.
470	Destroy	Destroys the stringlist.
471	Exchange	Implements the TStrings.Exchange (484) function.
472	Find	Locates the index for a given string in sorted lists.
472	IndexOf	Overrides the TStrings.IndexOf (486) property.
472	Insert	Overrides the TStrings.Insert (487) method.
473	Sort	Sorts the strings in the list.

4.72.3 Property overview

Page	Properties	Access	Description
474	CaseSensitive	rw	Indicates whether comparing strings happens in a case sensitive manner.
473	Duplicates	rw	Describes the behaviour of a sorted list with respect to duplicate strings.
474	OnChange	rw	Event triggered after the list was modified.
475	OnChanging	rw	Event triggered when the list is about to be modified.
475	OwnsObjects	rw	Determines whether the stringlist owns it's objects or not.
474	Sorted	rw	Determines whether the list is sorted or not.
475	SortStyle	rw	Sort style for strings.

4.72.4 TStringList.Create

Declaration:
```
constructor Create
constructor Create(anOwnsObjects: Boolean)
```

Visibility: public

4.72.5 TStringList.Destroy

Synopsis: Destroys the stringlist.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` clears the stringlist, release all memory allocated for the storage of the strings, and then calls the inherited destroy method.

Remark Any objects associated to strings in the list will *not* be destroyed; it is the responsibility of the caller to destroy all objects associated with strings in the list.

4.72.6 TStringList.Add

Synopsis: Implements the TStrings.Add (480) function.

Declaration: `function Add(const S: string) : Integer;  Override`

Visibility: public

Description: `Add` will add `S` to the list. If the list is sorted and the string `S` is already present in the list and TStringList.Duplicates (473) is `dupError` then an EStringListError (312) exception is raised. If `Duplicates` is set to `dupIgnore` then the return value is the index of the previous entry.

If the list is sorted, new strings will not necessarily be added to the end of the list, rather they will be inserted at their alphabetical position.

Errors: If the list is sorted and the string `S` is already present in the list and TStringList.Duplicates (473) is `dupError` then an EStringListError (312) exception is raised.

See also: TStringList.Insert (472), TStringList.Duplicates (473)

4.72.7 TStringList.Clear

Synopsis: Implements the TStrings.Clear (483) function.

Declaration: `procedure Clear;  Override`

Visibility: public

Description: Implements the TStrings.Clear (483) function.

4.72.8 TStringList.Delete

Synopsis: Implements the TStrings.Delete (483) function.

Declaration: `procedure Delete(Index: Integer);  Override`

Visibility: public

Description: Implements the TStrings.Delete (483) function.

4.72.9 TStringList.Exchange

Synopsis: Implements the TStrings.Exchange (484) function.

Declaration: `procedure Exchange(Index1: Integer; Index2: Integer);  Override`

Visibility: public

Description: `Exchange` will exchange two items in the list as described in TStrings.Exchange (484).

Remark `Exchange` will not check whether the list is sorted or not; if `Exchange` is called on a sorted list and the strings are not identical, the sort order of the list will be destroyed.

See also: TStringList.Sorted (474), TStrings.Exchange (484)

4.72.10 TStringList.Find

Synopsis: Locates the index for a given string in sorted lists.

Declaration: `function Find(const S: string; out Index: Integer) : Boolean;  Virtual`

Visibility: public

Description: `Find` returns `True` if the string `S` is present in the list. Upon exit, the `Index` parameter will contain the position of the string in the list. If the string is not found, the function will return `False` and `Index` will contain the position where the string will be inserted if it is added to the list.

Remark

1.Use this method only on sorted lists. For unsorted lists, use TStringList.IndexOf (472) instead.

2.Find uses a binary search method to locate the string

4.72.11 TStringList.IndexOf

Synopsis: Overrides the TStrings.IndexOf (486) property.

Declaration: `function IndexOf(const S: string) : Integer;  Override`

Visibility: public

Description: `IndexOf` overrides the ancestor method TStrings.indexOf (486). It tries to optimize the search by executing a binary search if the list is sorted. The function returns the position of `S` if it is found in the list, or -1 if the string is not found in the list.

See also: TStrings.IndexOf (486), TStringList.Find (472)

4.72.12 TStringList.Insert

Synopsis: Overrides the TStrings.Insert (487) method.

Declaration: `procedure Insert(Index: Integer; const S: string);  Override`

Visibility: public

Description: `Insert` will insert the string `S` at position `Index` in the list. If the list is sorted, an EStringListError (312) exception will be raised instead. `Index` is a zero-based position.

Errors: If `Index` contains an invalid value (less than zero or larger than `Count`, or the list is sorted, an EStringListError (312) exception will be raised.

See also: TStringList.Add (471), TStrings.Insert (487), TStrings.InsertObject (488)

4.72.13 TStringList.Sort

Synopsis: Sorts the strings in the list.

Declaration: `procedure Sort;  Virtual`

Visibility: public

Description: `Sort` will sort the strings in the list using the quicksort algorithm. If the list has its TStringList.Sorted (474) property set to `True` then nothing will be done.

See also: TStringList.Sorted (474)

4.72.14 TStringList.CustomSort

Synopsis: Sort the stringlist using a custom sort algorithm.

Declaration: `procedure CustomSort(CompareFn: TStringListSortCompare);  Virtual`

Visibility: public

Description: `CustomSort` sorts the stringlist with a custom comparison function. The function should compare 2 elements in the list, and return a negative number if the first item is before the second. It should return 0 if the elements are equal, and a positive result indicates that the second elements should be before the first.

See also: TStringList.Sorted (474), TStringList.Sort (473)

4.72.15 TStringList.Duplicates

Synopsis: Describes the behaviour of a sorted list with respect to duplicate strings.

Declaration: `Property Duplicates : TDuplicates`

Visibility: public

Access: Read,Write

Description: `Duplicates` describes what to do in case a duplicate value is added to the list:

Table 4.32:

dupIgnore	Duplicate values will not be added to the list, but no error will be triggered.
dupError	If an attempt is made to add a duplicate value to the list, an EStringListError (312) exception is raised.
dupAccept	Duplicate values can be added to the list.

The default value is `dupAccept`.

If the stringlist is not sorted, the `Duplicates` setting is ignored.

4.72.16 TStringList.Sorted

Synopsis: Determines whether the list is sorted or not.

Declaration: `Property Sorted : Boolean`

Visibility: public

Access: Read,Write

Description: `Sorted` can be set to `True` in order to cause the list of strings to be sorted. Further additions to the list will be inserted at the correct position so the list remains sorted at all times. Setting the property to `False` has no immediate effect, but will allow strings to be inserted at any position.

Remark

1. When `Sorted` is `True`, TStringList.Insert (472) cannot be used. For sorted lists, TStringList.Add (471) should be used instead.
2. If `Sorted` is `True`, the TStringList.Duplicates (473) setting has effect. This setting is ignored when `Sorted` is `False`.

See also: TStringList.Sort (473), TStringList.Duplicates (473), TStringList.Add (471), TstringList.Insert (472)

4.72.17 TStringList.CaseSensitive

Synopsis: Indicates whether comparing strings happens in a case sensitive manner.

Declaration: `Property CaseSensitive : Boolean`

Visibility: public

Access: Read,Write

Description: `CaseSensitive` indicates how string values are compared: When `True` this happens case sensitively, and when `False`, the comparison is done in a case insensitive manner.

This property influences IndexOf (486), Find (472), IndexOfName (487) and Sort (473).

See also: IndexOf (486), Find (472), IndexOfName (487), Sort (473)

4.72.18 TStringList.OnChange

Synopsis: Event triggered after the list was modified.

Declaration: `Property OnChange : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnChange` can be assigned to respond to changes that have occurred in the list. The handler is called whenever strings are added, moved, modified or deleted from the list.

The `Onchange` event is triggered after the modification took place. When the modification is about to happen, an TstringList.OnChanging (475) event occurs.

See also: TStringList.OnChanging (475)

4.72.19 TStringList.OnChanging

Synopsis: Event triggered when the list is about to be modified.

Declaration: `Property OnChanging : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnChanging` can be assigned to respond to changes that will occurred in the list. The handler is called whenever strings will be added, moved, modified or deleted from the list.

The `Onchanging` event is triggered before the modification will take place. When the modification has happened, an TstringList.OnChange (474) event occurs.

See also: TStringList.OnChange (474)

4.72.20 TStringList.OwnsObjects

Synopsis: Determines whether the stringlist owns it's objects or not.

Declaration: `Property OwnsObjects : Boolean`

Visibility: public

Access: Read,Write

Description: `OwnsObjects` can be set to `true` to let the stringlist instance own the objects in the list: if an element is removed from the list, the associated object (if there is any) will be freed as well. The same is true if the list is cleared or destroyed.

See also: TStrings.Objects (496)

4.72.21 TStringList.SortStyle

Synopsis: Sort style for strings.

Declaration: `Property SortStyle : TStringsSortStyle`

Visibility: public

Access: Read,Write

Description: `SortStyle` sets the sort style for the `TStringList` instance. Setting it to `sslAuto` will keep the list sorted automatically at all times.

Find (472) (and hence IndexOf (486)) will use a binary search if `SortStyle` differs from `sslNone`. if `SortStyle` is `sslNone`, then it will raise an exception.

See also: TStringsSortStyle (289), TStrings.Sorted (475)

4.73 TStrings

4.73.1 Description

`TStrings` implements an abstract class to manage an array of strings. It introduces methods to set and retrieve strings in the array, searching for a particular string, concatenating the strings and so on. It also allows an arbitrary object to be associated with each string.

It also introduces methods to manage a series of `name=value` settings, as found in many configuration files.

An instance of `TStrings` is never created directly, instead a descendant class such as TStringList (470) should be created. This is because `TStrings` is an abstract class which does not implement all methods; `TStrings` also doesn't store any strings, this is the functionality introduced in descendants such as TStringList (470).

`TStrings` implements the IFPObserved (314) interface: when the stringlist is changed, a `ooChanged` notification is sent to all observers.

See also: TStringList (470), IFPObserved (314)

4.73.2 Method overview

Page	Method	Description
480	Add	Add a string to the list.
481	AddCommaText	Adds a comma-separated list of strings.
482	AddDelimitedtext	Adds a separator-delimited list of strings.
480	AddObject	Add a string and associated object to the list.
480	AddPair	Add a name-value pair.
481	AddStrings	Add contents of another stringlist to this list.
481	AddText	Add text to the string list.
482	Append	Add a string to the list.
482	Assign	Assign the contents of another stringlist to this one.
483	BeginUpdate	Mark the beginning of an update batch.
483	Clear	Removes all strings and associated objects from the list.
478	Create	Initializ a new `TStrings` instance.
483	Delete	Delete a string from the list.
479	Destroy	Frees all strings and objects, and removes the list from memory.
484	EndUpdate	Mark the end of an update batch.
484	Equals	Compares the contents of two stringlists.
484	Exchange	Exchanges two strings in the list.
485	ExtractName	Extract the name part of a string.
485	Fill	Set a range of strings in the list to the specified value.
485	Filter	Return a filtered list.
485	ForEach	Call a callback for each string in the list.
486	GetEnumerator	Create an `IEnumerator` instance.
486	GetNameValue	Return both name and value of a name,value pair based on it's index.
486	GetText	Returns the contents as a PChar.
486	IndexOf	Find a string in the list and return its position.
487	IndexOfName	Finds the index of a name in the name-value pairs.
487	IndexOfObject	Finds an object in the list and returns its index.
487	Insert	Insert a string in the list.
488	InsertObject	Insert a string and associated object in the list.
488	LastIndexOf	Return the index of the last occurrence of a string.
488	LoadFromFile	Load the contents of a file as a series of strings.
489	LoadFromStream	Load the contents of a stream as a series of strings.
489	Map	Call a map routine for each string in the list.
489	Move	Move a string from one place in the list to another.
490	Pop	Remove the last string of the list and return it.
490	Reduce	Get a reduced value from the string list.
490	Reverse	Reverse the order of the strings in the list.
491	SaveToFile	Save the contents of the list to a file.
491	SaveToStream	Save the contents of the string to a stream.
481	SetStrings	Replace the strings with the passed strings.
492	SetText	Set the contents of the list from a PChar.
491	Shift	Return and remove the first string in the list.
492	Slice	Return a subrange of strings starting at a given index.
479	ToObjectArray	Return the objects as an array of object.
479	ToStringArray	Return the strings as an array of string.

4.73.3 Property overview

Page	Properties	Access	Description
492	AlwaysQuote	rw	Always quote strings in DelimitedText.
492	Capacity	rw	Capacity of the list, i.e. number of strings that the list can currently hold before it tries to expand.
493	CommaText	rw	Contents of the list as a comma-separated string.
493	Count	r	Number of strings in the list.
494	DefaultEncoding	rw	Default encoding of stringlist.
494	DelimitedText	rw	Get or set all strings in the list in a delimited form.
495	Delimiter	rw	Delimiter character used in DelimitedText (494).
495	Encoding	r	Current encoding of stringlist.
495	LineBreak	rw	LineBreak character to use.
495	MissingNameValueSeparatorAction	rw	What to do if `NameValueSeparator` is missing in a string.
496	Names	r	Name parts of the name-value pairs in the list.
496	NameValueSeparator	rw	Value of the character used to separate name,value pairs.
496	Objects	rw	Indexed access to the objects associated with the strings in the list.
497	Options	rw	A set of TStringsOption (288) - various boolean properties.
497	QuoteChar	rw	Quote character used in DelimitedText (494).
497	SkipLastLineBreak	rw	Do not add a linebreak to the last item.
498	StrictDelimiter	rw	Should only the delimiter character be considered a delimiter.
498	Strings	rw	Indexed access to the strings in the list.
498	StringsAdapter	rw	Not implemented in Free Pascal.
499	Text	rw	Contents of the list as one big string.
499	TextLineBreakStyle	rw	Determines which line breaks to use in the Text (499) property.
497	TrailingLineBreak	rw	Add a linebreak to the last item.
499	UseLocale	rw	Determines what methods are used in strings comparison.
500	ValueFromIndex	rw	Return the value part of a string based on it's index.
500	Values	rw	Value parts of the name-value pairs in the list.
500	WriteBOM	rw	Write BOM when writing stringlist to stream.

4.73.4 TStrings.Create

Synopsis: Initializ a new `TStrings` instance.

Declaration: `constructor Create`

Visibility: public

Description: `Create` initializes the codepage variables of a new `TStrings` instance. Note that `TStrings` is an abstract class, you must always instantiate a descendent such as TStringList (470)

Errors: Only an EOutOfMemory exception can occur.

See also: TStringList (470)

4.73.5 TStrings.Destroy

Synopsis: Frees all strings and objects, and removes the list from memory.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` is the destructor of `TStrings` it does nothing except calling the inherited destructor.

4.73.6 TStrings.ToObjectArray

Synopsis: Return the objects as an array of object.

Declaration:
```
function ToObjectArray(aStart: Integer; aEnd: Integer) : TObjectDynArray
                        ;  Overload
function ToObjectArray : TObjectDynArray;  Overload
```

Visibility: public

Description: `ToObjectArray` returns all the values in the Objects (496) array as a dynamic array of `TObject` instances. If `aStart` and `aEnd` are specified, only the objects with indexes in the specified array (`aStart` and `aEnd` included) are returned. `Nil` values will also be returned.

Errors: If `aStart` is larger than `aEnd`, no elements are returned.

See also: TStrings.ToStringArray (479), TStrings.Strings (498), TStrings.Objects (496)

4.73.7 TStrings.ToStringArray

Synopsis: Return the strings as an array of string.

Declaration:
```
function ToStringArray(aStart: Integer; aEnd: Integer) : TStringDynArray
                        ;  Overload
function ToStringArray : TStringDynArray;  Overload
```

Visibility: public

Description: `ToObjectArray` returns all the values in the Objects (496) array as a dynamic array of `TObject` instances. If `aStart` and `aEnd` are specified, only the objects with indexes in the specified array (`aStart` and `aEnd` included) are returned. `Nil` values will also be returned.

Errors: If `aStart` is larger than `aEnd`, no elements are returned.

See also: TStrings.ToObjectArray (479), TStrings.Strings (498), TStrings.Objects (496)

4.73.8 TStrings.Add

Synopsis: Add a string to the list.

Declaration:
```
function Add(const S: string) : Integer;  Virtual;  Overload
function Add(const Fmt: string; const Args: Array of const) : Integer
            ;  Overload
```

Visibility: public

Description: `Add` adds `S` at the end of the list and returns the index of `S` in the list (which should equal Tstrings.Count (493)

See also: TStrings.Objects (496), TStrings.AddObject (480), TStrings.Insert (487), TStrings.Delete (483), TStrings.Strings (498), TStrings.Count (493)

4.73.9 TStrings.AddObject

Synopsis: Add a string and associated object to the list.

Declaration:
```
function AddObject(const S: string; AObject: TObject) : Integer
                   ;  Virtual;  Overload
function AddObject(const Fmt: string; Args: Array of const;
                   AObject: TObject) : Integer;  Overload
```

Visibility: public

Description: `AddObject` adds `S` to the list of strings, and associates `AObject` with it. It returns the index of `S`.

Remark An object added to the list is not automatically destroyed by the list when the list is destroyed or the string it is associated with is deleted. It is the responsibility of the application to destroy any objects associated with strings.

See also: TStrings.Add (480), TStrings.Strings (498), TStrings.Objects (496), TStrings.InsertObject (488)

4.73.10 TStrings.AddPair

Synopsis: Add a name-value pair.

Declaration:
```
function AddPair(const AName: string; const AValue: string) : TStrings
                 ;  Overload
function AddPair(const AName: string; const AValue: string;
                 AObject: TObject) : TStrings;  Overload
```

Visibility: public

Description: `AddPair` adds a `Name=Value` pair with the `AName` and `AValue` parameters, optionally with object `AObject`. It uses the correct NameValueSeparator (272) character to do so.

`AddPair` does not test whether `aName` already exists, so it is possible to add duplicate names.

Errors: None.

See also: TStrings.GetNameValue (486), TStrings.NameValueSeparator (496), TStrings.Add (480), TStrings.AddObject (480)

4.73.11 TStrings.AddStrings

Synopsis: Add contents of another stringlist to this list.

Declaration:
```
procedure AddStrings(TheStrings: TStrings);  Virtual;  Overload
procedure AddStrings(TheStrings: TStrings; ClearFirst: Boolean)
                    ;  Overload
procedure AddStrings(const TheStrings: Array of string);  Virtual
                    ;  Overload
procedure AddStrings(const TheStrings: Array of string;
                    ClearFirst: Boolean);  Overload
```

Visibility: public

Description: `AddStrings` adds the contents of `TheStrings` to the stringlist. Any associated objects are added as well.

See also: TStrings.Add (480), TStrings.Assign (482)

4.73.12 TStrings.SetStrings

Synopsis: Replace the strings with the passed strings.

Declaration:
```
procedure SetStrings(TheStrings: TStrings);  Virtual;  Overload
procedure SetStrings(TheStrings: Array of string);  Virtual;  Overload
```

Visibility: public

Description: `SetStrings` sets the strings of the stringlist to the specified values in `TheStrings`. The values can be specified as another stringlist or as an array of strings. In difference with the Assign (482) method it does not copy all other properties.

Internally it clears the array and calls TStrings.AddStrings (481).

See also: TStrings.Assign (482), TStrings.AddStrings (481)

4.73.13 TStrings.AddText

Synopsis: Add text to the string list.

Declaration:
```
procedure AddText(const S: string);  Virtual
```

Visibility: public

Description: `AddText` adds `S` to the strings. It is identical in function to setting Text (272) but does not clear the list of strings first: `S` is split into lines, and each line is added to the list.

See also: TString.Text (272)

4.73.14 TStrings.AddCommaText

Synopsis: Adds a comma-separated list of strings.

Declaration:
```
procedure AddCommaText(const S: string)
```

Visibility: public

Description: `AddCommaText` parses `S` using the same mechanism as when setting TStrings.CommaText (493) but adds the values instead of removing existing values.

`AddCommaText` also observes the TStrings.StrictDelimiter (498) and TStrings.QuoteChar (497) properties.

See also: TStrings.AddDelimitedText (482), TStrings.CommaText (493), TStrings.QuoteChar (497), TStrings.StrictDelimiter (498)

4.73.15 TStrings.AddDelimitedtext

Synopsis: Adds a separator-delimited list of strings.

Declaration:
```
procedure AddDelimitedText(const S: string; ADelimiter: Char;
                           AStrictDelimiter: Boolean);  Overload
procedure AddDelimitedtext(const S: string);  Overload
```

Visibility: public

Description: `AddDelimitedText` parses `S` using the same mechanism as when setting TStrings.DelimitedText (494) but adds the values instead of removing existing values. In difference with TStrings.AddCommaText (481) it uses the TStrings.Delimiter (495) character to separate the values.

`AddDelimitedText` also observes the TStrings.Delimiter (495), TStrings.StrictDelimiter (498) and TStrings.QuoteChar (497) properties.

See also: TStrings.AddCommaText (481), TStrings.DelimitedText (494), TStrings.Delimiter (495), TStrings.StrictDelimiter (498), TStrings.QuoteChar (497)

4.73.16 TStrings.Append

Synopsis: Add a string to the list.

Declaration:
```
procedure Append(const S: string)
```

Visibility: public

Description: `Append` does the same as TStrings.Add (480), only it does not return the index of the inserted string.

See also: TStrings.Add (480)

4.73.17 TStrings.Assign

Synopsis: Assign the contents of another stringlist to this one.

Declaration:
```
procedure Assign(Source: TPersistent);  Override
```

Visibility: public

Description: `Assign` replaces the contents of the stringlist with the contents of `Source` if `Source` is also of type `TStrings`. Any associated objects are copied as well.

See also: TStrings.Add (480), TStrings.AddStrings (481), TPersistent.Assign (436)

4.73.18 TStrings.BeginUpdate

Synopsis: Mark the beginning of an update batch.

Declaration: `procedure BeginUpdate`

Visibility: public

Description: `BeginUpdate` increases the update count by one. It is advisable to call `BeginUpdate` before lengthy operations on the stringlist. At the end of these operation, TStrings.EndUpdate (484) should be called to mark the end of the operation. Descendant classes may use this information to perform optimizations. e.g. updating the screen only once after many strings were added to the list.

All `TStrings` methods that modify the string list call `BeginUpdate` before the actual operation, and call `endUpdate` when the operation is finished. Descendant classes should also call these methods when modifying the string list.

Remark Always put the corresponding call to TStrings.EndUpdate (484) in the context of a `Finally` block, to ensure that the update count is always decreased at the end of the operation, even if an exception occurred:

```
With MyStrings do
    try
      BeginUpdate;
      // Some lengthy operation.
    Finally
      EndUpdate
    end;
```

See also: TStrings.EndUpdate (484)

4.73.19 TStrings.Clear

Synopsis: Removes all strings and associated objects from the list.

Declaration: `procedure Clear;  Virtual;  Abstract`

Visibility: public

Description: `Clear` will remove all strings and their associated objects from the list. After a call to clear, TStrings.Count (493) is zero.

Since it is an abstract method, `TStrings` itself does not implement `Clear`. Descendant classes such as TStringList (470) implement this method.

See also: TStrings.Objects (496), TStrings.Strings (498), TStrings.Delete (483), TStrings.Count (493)

4.73.20 TStrings.Delete

Synopsis: Delete a string from the list.

Declaration: `procedure Delete(Index: Integer);  Virtual;  Abstract`

Visibility: public

Description: `Delete` deletes the string at position `Index` from the list. The associated object is also removed from the list, but not destroyed. `Index` is zero-based, and should be in the range `0` to `Count-1`.

Since it is an abstract method, `TStrings` itself does not implement `Delete`. Descendant classes such as TStringList (470) implement this method.

Errors: If `Index` is not in the allowed range, an EStringListError (312) is raised.

See also: TStrings.Insert (487), TStrings.Objects (496), TStrings.Strings (498), TStrings.Clear (483)

4.73.21 TStrings.EndUpdate

Synopsis: Mark the end of an update batch.

Declaration:
```
procedure EndUpdate
```

Visibility: public

Description: `EndUpdate` should be called at the end of a lengthy operation on the stringlist, but only if there was a call to `BeginUpdate` before the operation was started. It is best to put the call to `EndUpdate` in the context of a `Finally` block, so it will be called even if an exception occurs.

For more information, see TStrings.BeginUpdate (483).

`TStrings` implements the IFPObserved (314) interface: when `EndUpdate` is called, a `ooChanged` notification is sent to all observers.

See also: TStrings.BeginUpdate (483), IFPObserved (314)

4.73.22 TStrings.Equals

Synopsis: Compares the contents of two stringlists.

Declaration:
```
function Equals(Obj: TObject) : Boolean;  Override;  Overload
function Equals(TheStrings: TStrings) : Boolean;  Overload
```

Visibility: public

Description: `Equals` compares the contents of the stringlist with the contents of `TheStrings`. If the contents match, i.e. the stringlist contain an equal amount of strings, and all strings match, then `True` is returned. If the number of strings in the lists is unequal, or they contain one or more different strings, `False` is returned.

Remark

1. The strings are compared case-insensitively.
2. The associated objects are not compared

See also: TStrings.Objects (496), TStrings.Strings (498), TStrings.Count (493), TStrings.Assign (482)

4.73.23 TStrings.Exchange

Synopsis: Exchanges two strings in the list.

Declaration:
```
procedure Exchange(Index1: Integer; Index2: Integer);  Virtual
```

Visibility: public

Description: `Exchange` exchanges the strings at positions `Index1` and `Index2`. The associated objects are also exchanged.

Both indexes must be in the range of valid indexes, i.e. must have a value between `0` and `Count-1`.

Errors: If either `Index1` or `Index2` is not in the range of valid indexes, an EStringListError (312) exception is raised.

See also: TStrings.Move (489), TStrings.Strings (498), TStrings.Count (493)

4.73.24 TStrings.ExtractName

Synopsis: Extract the name part of a string.

Declaration:
```
function ExtractName(const S: string) : string
```

Visibility: public

Description: `ExtractName` returns the name part (the part before the NameValueSeparator (496) character) of the string. If the character is not present, an empty string is returned. The resulting string is not trimmed, it can end or start with spaces.

See also: NameValueSeparator (496)

4.73.25 TStrings.Filter

Synopsis: Return a filtered list.

Declaration:
```
procedure Filter(aFilter: TStringsFilterMethod; aList: TStrings)
function Filter(aFilter: TStringsFilterMethod) : TStrings
```

Visibility: public

Description: `Filter` applies `aFilter` to each string in the list, and if `aFilter` returns `True`, adds the string to the `aList` list. If the `aList` argument is not supplied, the result is a TStringList (470) instance that must be freed by the caller.

See also: TStringsFilterMethod (287), TStringList (470), TStrings.Foreach (485), TStrings.Map (489), TStrings.Reduce (490), TStrings.Reverse (490)

4.73.26 TStrings.Fill

Synopsis: Set a range of strings in the list to the specified value.

Declaration:
```
procedure Fill(const aValue: string; aStart: Integer; aEnd: Integer)
```

Visibility: public

Description: `Fill` sets the strings with indexes in the range `aStart`..`aEnd` to `aValue`. if the `aStart`..`aEnd` are outside the valid range of indexes for the list, their values are adapted to fit the range. (0 to Count (493)-1)

See also: Count (493), Strings (498)

4.73.27 TStrings.ForEach

Synopsis: Call a callback for each string in the list.

Declaration:
```
procedure ForEach(aCallback: TStringsForEachMethod)
procedure ForEach(aCallback: TStringsForEachMethodEx)
procedure ForEach(aCallback: TStringsForEachMethodExObj)
```

Visibility: public

Description: `Foreach` calls `aCallback` for every string in the list. It passes the string and optionally the string index and associated object to the callback.

See also: TStrings.Filter (485), TStrings.Map (489), TStrings.Fill (485), TStringsForEachMethod (287), TStrings-ForEachMethodEx (288), TStringsForEachMethodExObj (288), TStrings.Reverse (490)

4.73.28 TStrings.GetEnumerator

Synopsis: Create an `IEnumerator` instance.

Declaration:
```
function GetEnumerator : TStringsEnumerator
```

Visibility: public

Description: `GetEnumerator` is the implementation of the IEnumerable (1616) interface for `TStrings`. It creates a TStringsEnumerator (501) instance and returns it's IEnumerator (1617) interface.

See also: TStringsEnumerator (501), IEnumerator (1617), IEnumerable (1616)

4.73.29 TStrings.GetNameValue

Synopsis: Return both name and value of a name,value pair based on it's index.

Declaration:
```
procedure GetNameValue(Index: Integer; out AName: string;
                       out AValue: string)
```

Visibility: public

Description: Return both name and value of a name,value pair based on it's index.

4.73.30 TStrings.GetText

Synopsis: Returns the contents as a PChar.

Declaration:
```
function GetText : PChar;  Virtual
```

Visibility: public

Description: `GetText` allocates a memory buffer and copies the contents of the stringlist to this buffer as a series of strings, separated by an end-of-line marker. The buffer is zero terminated.

Remark The caller is responsible for freeing the returned memory buffer.

4.73.31 TStrings.IndexOf

Synopsis: Find a string in the list and return its position.

Declaration:
```
function IndexOf(const S: string) : Integer;  Virtual
function IndexOf(const S: string; aStart: Integer) : Integer;  Virtual
```

Visibility: public

Description: `IndexOf` searches the list for `S` starting from index `aStart`. The start position `aStart` is 0 if not specified. If it is negative, it is considered an offset from TStrings.Count (493). The search is case-insensitive or case sensitive depending on the value of the CaseSensitive (**??**) . If a matching entry is found, its position is returned. if no matching string is found, `-1` is returned.

Remark

1.Only the first occurrence of the string is returned.

2.The returned position is zero-based, i.e. 0 indicates the first string in the list.

See also: TStrings.IndexOfObject (487), TStrings.IndexOfName (487), TStrings.Strings (498), TStrings.LastIndexOf (488), TStrings.Count (493)

4.73.32 TStrings.IndexOfName

Synopsis: Finds the index of a name in the name-value pairs.

Declaration: `function IndexOfName(const Name: string) : Integer; Virtual`

Visibility: public

Description: `IndexOfName` searches in the list of strings for a name-value pair with name part `Name`. If such a pair is found, it returns the index of the pair in the stringlist. If no such pair is found, the function returns `-1`. The search is done case-insensitive.

Remark

1. Only the first occurrence of a matching name-value pair is returned.
2. The returned position is zero-based, i.e. 0 indicates the first string in the list.

See also: TStrings.IndexOf (486), TStrings.IndexOfObject (487), TStrings.Strings (498), TStrings.LastIndexOf (488)

4.73.33 TStrings.IndexOfObject

Synopsis: Finds an object in the list and returns its index.

Declaration: `function IndexOfObject(AObject: TObject) : Integer; Virtual`

Visibility: public

Description: `IndexOfObject` searches through the list of strings till it find a string associated with `AObject`, and returns the index of this string. If no such string is found, `-1` is returned.

Remark

1. Only the first occurrence of a string with associated object `AObject` is returned; if more strings in the list can be associated with `AObject`, they will not be found by this routine.
2. The returned position is zero-based, i.e. 0 indicates the first string in the list.

See also: TStrings.LastIndexOf (488), TStrings.IndexOf (486), TStrings.Objects (496)

4.73.34 TStrings.Insert

Synopsis: Insert a string in the list.

Declaration: `procedure Insert(Index: Integer; const S: string); Virtual; Abstract`

Visibility: public

Description: `Insert` inserts the string `S` at position `Index` in the list. `Index` is a zero-based position, and can have values from `0` to `Count`. If `Index` equals `Count` then the string is appended to the list.

Remark

1. All methods that add strings to the list use `Insert` to add a string to the list.
2. If the string has an associated object, use TStrings.InsertObject (488) instead.

Errors: If `Index` is less than zero or larger than `Count` then an EStringListError (312) exception is raised.

See also: TStrings.Add (480), TStrings.InsertObject (488), TStrings.Append (482), TStrings.Delete (483)

4.73.35 TStrings.InsertObject

Synopsis: Insert a string and associated object in the list.

Declaration:
```
procedure InsertObject(Index: Integer; const S: string;
                       AObject: TObject)
```

Visibility: public

Description: `InsertObject` inserts the string `S` and its associated object`AObject` at position `Index` in the list. `Index` is a zero-based position, and can have values from `0` to `Count`. If `Index` equals `Count` then the string is appended to the list.

Errors: If `Index` is less than zero or larger than `Count` then an EStringListError (312) exception is raised.

See also: TStrings.Insert (487), TStrings.AddObject (480), TStrings.Append (482), TStrings.Delete (483)

4.73.36 TStrings.LastIndexOf

Synopsis: Return the index of the last occurrence of a string.

Declaration:
```
function LastIndexOf(const S: string; aStart: Integer) : Integer
                     ;  Virtual
function LastIndexOf(const S: string) : Integer
```

Visibility: public

Description: `LastIndexOf` returns the index of the last occurrence of the string `S`. If `aStart` is specified, the search starts at the specified index in the list (it defaults to `Count-1`). If `aStart` is negative, it is used as an offset to Count (493), i.e. `aCount` equal to -1 will start the search at `Count-1`.

If `aStart` is larger than `Count-1` it will be capped to `Count-1`.

See also: IndexOf (486), IndexOfObject (487)

4.73.37 TStrings.LoadFromFile

Synopsis: Load the contents of a file as a series of strings.

Declaration:
```
procedure LoadFromFile(const FileName: string);  Virtual;  Overload
procedure LoadFromFile(const FileName: string; IgnoreEncoding: Boolean)
procedure LoadFromFile(const FileName: string; AEncoding: TEncoding)
                       ;  Virtual;  Overload
```

Visibility: public

Description: `LoadFromFile` loads the contents of a file into the stringlist. Each line in the file (as marked by the end-of-line marker of the particular OS the application runs on) becomes one string in the stringlist. This action replaces the contents of the stringlist, it does not append the strings to the current content.

`LoadFromFile` simply creates a file stream (395) with the given filename, and then executes TStrings.LoadfromStream (489); after that the file stream object is destroyed again.

See also: TStrings.LoadFromStream (489), TStrings.SaveToFile (491), Tstrings.SaveToStream (491)

4.73.38 TStrings.LoadFromStream

Synopsis: Load the contents of a stream as a series of strings.

Declaration:
```
procedure LoadFromStream(Stream: TStream);  Virtual;  Overload
procedure LoadFromStream(Stream: TStream; IgnoreEncoding: Boolean)
                         ;  Overload
procedure LoadFromStream(Stream: TStream; AEncoding: TEncoding)
                         ;  Virtual;  Overload
```

Visibility: public

Description: `LoadFromStream` loads the contents of `Stream` into the stringlist. Each line in the stream (as marked by the end-of-line marker of the particular OS the application runs on) becomes one string in the stringlist. This action replaces the contents of the stringlist, it does not append the strings to the current content.

See also: TStrings.LoadFromFile (488), TStrings.SaveToFile (491), Tstrings.SaveToStream (491)

4.73.39 TStrings.Map

Synopsis: Call a map routine for each string in the list.

Declaration:
```
procedure Map(aMap: TStringsMapMethod; aList: TStrings)
function Map(aMap: TStringsMapMethod) : TStrings
```

Visibility: public

Description: `Map` applies `aMap` to each string in the list and adds the result to the list `aList`. If `aList` is not specified, the result is a TStringList (470) instance which must be freed by the caller.

See also: TStrings.ForEach (485), TStrings.Map (489), TStrings.Filter (485), TStringsMapMethod (288), TStrings.Reverse (490)

4.73.40 TStrings.Move

Synopsis: Move a string from one place in the list to another.

Declaration:
```
procedure Move(CurIndex: Integer; NewIndex: Integer);  Virtual
```

Visibility: public

Description: `Move` moves the string at position `CurIndex` so it has position `NewIndex` after the move operation. The object associated to the string is also moved. `CurIndex` and `NewIndex` should be in the range of `0` to `Count-1`.

Remark `NewIndex` is *not* the position in the stringlist before the move operation starts. The move operation

1. removes the string from position `CurIndex`
2. inserts the string at position `NewIndex`

This may not lead to the desired result if `NewIndex` is bigger than `CurIndex`. Consider the following example:

```
With MyStrings do
    begin
    Clear;
    Add('String 0');
```

```
Add('String 1');
Add('String 2');
Add('String 3');
Add('String 4');
Move(1,3);
end;
```

After the `Move` operation has completed, 'String 1' will be between 'String 3' and 'String 4'.

Errors: If either `CurIndex` or `NewIndex` is outside the allowed range, an EStringListError (312) is raised.

See also: TStrings.Exchange (484)

4.73.41 TStrings.Pop

Synopsis: Remove the last string of the list and return it.

Declaration:
```
function Pop : string
```

Visibility: public

Description: `Pop` returns the last list in the string and removes it from the list. If there are no strings (i.e. Count (493) equals zero), an empty string is returned.

See also: TStrings.Add (480), TStrings.Shift (491), TStrings.Slice (492)

4.73.42 TStrings.Reduce

Synopsis: Get a reduced value from the string list.

Declaration:
```
function Reduce(aReduceMethod: TStringsReduceMethod;
                const startingValue: string) : string
```

Visibility: public

Description: `Reduce` iterates over all strings in the list and calls the `aReduceMethod` callback with 2 strings: the result of the previous iteration and the current string. The return value will be used in the next iteration, or will result in the end result of the method if the last string was reached.

See also: TStrings.Filter (485), TStrings.Map (489), TStrings.Foreach (485), TStringsReduceMethod (289), TStrings.Reverse (490)

4.73.43 TStrings.Reverse

Synopsis: Reverse the order of the strings in the list.

Declaration:
```
function Reverse : TStrings
procedure Reverse(aList: TStrings)
```

Visibility: public

Description: `Reverse` reverses the order of the strings in the list.

See also: TStrings.Filter (485), TStrings.Map (489), TStrings.Foreach (485), TStrings.Reduce (490), TStrings.Sort (475)

4.73.44 TStrings.SaveToFile

Synopsis: Save the contents of the list to a file.

Declaration:
```
procedure SaveToFile(const FileName: string);  Virtual;  Overload
procedure SaveToFile(const FileName: string; IgnoreEncoding: Boolean)
                    ;  Overload
procedure SaveToFile(const FileName: string; AEncoding: TEncoding)
                    ;  Virtual;  Overload
```

Visibility: public

Description: `SaveToFile` saves the contents of the stringlist to the file with name `FileName`. It writes the strings to the file, separated by end-of-line markers, so each line in the file will contain 1 string from the stringlist.

`SaveToFile` creates a file stream (395) with name `FileName`, calls TStrings.SaveToStream (491) and then destroys the file stream object.

Errors: An EStreamError (312) exception can be raised if the file `FileName` cannot be opened, or if it cannot be written to.

See also: TStrings.SaveToStream (491), Tstrings.LoadFromStream (489), TStrings.LoadFromFile (488)

4.73.45 TStrings.SaveToStream

Synopsis: Save the contents of the string to a stream.

Declaration:
```
procedure SaveToStream(Stream: TStream);  Virtual;  Overload
procedure SaveToStream(Stream: TStream; IgnoreEncoding: Boolean)
                      ;  Overload
procedure SaveToStream(Stream: TStream; AEncoding: TEncoding);  Virtual
                      ;  Overload
```

Visibility: public

Description: `SaveToStream` saves the contents of the stringlist to `Stream`. It writes the strings to the stream, separated by end-of-line markers, so each 'line' in the stream will contain 1 string from the stringlist.

Errors: An EStreamError (312) exception can be raised if the stream cannot be written to.

See also: TStrings.SaveToFile (491), Tstrings.LoadFromStream (489), TStrings.LoadFromFile (488)

4.73.46 TStrings.Shift

Synopsis: Return and remove the first string in the list.

Declaration:
```
function Shift : string
```

Visibility: public

Description: `Shift` returns the first string in the list and removes it from the list, causing all values to shift one position. If there are no strings in the list, an empty value is returned.

See also: TStrings.Slice (492), TStrings.Pop (490), TStrings.Delete (483), TStrings.Insert (487)

4.73.47 TStrings.Slice

Synopsis: Return a subrange of strings starting at a given index.

Declaration:
```
procedure Slice(fromIndex: Integer; aList: TStrings)
function Slice(fromIndex: Integer) : TStrings
```

Visibility: public

Description: `Slice` adds to `aList` the strings in the current list, starting with the string at index `FromIndex`. If `aList` is not specified, a TStringList (470) instance is created and used. The caller must free the new instance.

See also: TStrings.Filter (485), TStrings.Shift (491), TStrings.Pop (490), TStrings.Map (489), TStrings.Delete (483), TStrings.Insert (487)

4.73.48 TStrings.SetText

Synopsis: Set the contents of the list from a PChar.

Declaration:
```
procedure SetText(TheText: PChar);  Virtual
```

Visibility: public

Description: `SetText` parses the contents of `TheText` and fills the stringlist based on the contents. It regards `TheText` as a series of strings, separated by end-of-line markers. Each of these strings is added to the stringlist.

See also: TStrings.Text (499)

4.73.49 TStrings.AlwaysQuote

Synopsis: Always quote strings in DelimitedText.

Declaration:
```
Property AlwaysQuote : Boolean
```

Visibility: public

Access: Read,Write

Description: `AlwaysQuote` tells the stringlist instance to quote strings in DelimitedText (494) . The default is to quote strings only when they have whitespace in them.

See also: DelimitedText (494), CommaText (494), StrictDelimiter (498)

4.73.50 TStrings.Capacity

Synopsis: Capacity of the list, i.e. number of strings that the list can currently hold before it tries to expand.

Declaration:
```
Property Capacity : Integer
```

Visibility: public

Access: Read,Write

Description: `Capacity` is the number of strings that the list can hold before it tries to allocate more memory.

`TStrings` returns TStrings.Count (493) when read. Trying to set the capacity has no effect. Descendant classes such as TStringlist (470) can override this property such that it actually sets the new capacity.

See also: TStringList (470), TStrings.Count (493)

4.73.51 TStrings.CommaText

Synopsis: Contents of the list as a comma-separated string.

Declaration: `Property CommaText : string`

Visibility: public

Access: Read,Write

Description: `CommaText` represents the stringlist as a single string, consisting of a comma-separated concatenation of the strings in the list. If one of the strings contains spaces, comma's or quotes it will be enclosed by double quotes. Any double quotes in a string will be doubled. For instance the following strings:

```
Comma,string
Quote"string
Space string
NormalSttring
```

is converted to

```
"Comma,string","Quote""String","Space string",NormalString
```

Conversely, when setting the `CommaText` property, the text will be parsed according to the rules outlined above, and the strings will be set accordingly. Note that spaces will in this context be regarded as string separators, unless the string as a whole is contained in double quotes. Spaces that occur next to a delimiter will be ignored. The following string:

```
"Comma,string" , "Quote""String",Space string,, NormalString
```

Will be converted to

```
Comma,String
Quote"String
Space
String

NormalString
```

This is a special case of the TStrings.DelimitedText (494) property where the quote character is always the double quote, and the delimiter is always the colon.

Setting `CommaText` is equivalent to clearing the stringlist and calling TStrings.AddCommaText (481).

See also: TStrings.Text (499), TStrings.SetText (492), TStrings.DelimitedText (494), TStrings.AddCommaText (481)

4.73.52 TStrings.Count

Synopsis: Number of strings in the list.

Declaration: `Property Count : Integer`

Visibility: public

Access: Read

Description: `Count` is the current number of strings in the list. `TStrings` does not implement this property; descendant classes should override the property read handler to return the correct value.

Strings in the list are always uniquely identified by their `Index`; the index of a string is zero-based, i.e. it's supported range is `0` to `Count-1`. trying to access a string with an index larger than or equal to `Count` will result in an error. Code that iterates over the list in a stringlist should always take into account the zero-based character of the list index.

See also: TStrings.Strings (498), TStrings.Objects (496), TStrings.Capacity (492)

4.73.53 TStrings.DefaultEncoding

Synopsis: Default encoding of stringlist.

Declaration: `Property DefaultEncoding : TEncoding`

Visibility: public

Access: Read,Write

Description: `DefaultEncoding` is the default encoding used by the `TStrings` instance. It is not the actual encoding, as specified by Encoding (495).

See also: Encoding (495)

4.73.54 TStrings.DelimitedText

Synopsis: Get or set all strings in the list in a delimited form.

Declaration: `Property DelimitedText : string`

Visibility: public

Access: Read,Write

Description: `DelimitedText` returns all strings, properly quoted with QuoteChar (497) and separated by the Delimiter (495) character.

Strings are quoted if they contain a space or any character with ASCII value less than 32.

The CommaText (493) property is a special case of delimited text where the delimiter character is a comma and the quote character is a double quote.

If StrictDelimiter (498) is set to `True`, then no quoting is done (The `QuoteChar` property is disregarded completely): the returned text will contain the items in the stringlist, separated by the `Delimiter` character. When writing the `DelimitedText` property, the text will be split at all occurrences of the `Delimiter` character; however, when reading, the `QuoteChar` property will be taken into account.

Setting TStrings.AddDelimitedText (482) is equivalent to clearing the list and calling TStrings.AddDelimitedText (482).

See also: TStrings.Delimiter (495), TStrings.StrictDelimiter (498), TStrings.Text (499), TStrings.QuoteChar (497), TStrings.CommaText (493), TStrings.AddDelimitedText (482)

4.73.55 TStrings.Delimiter

Synopsis: Delimiter character used in DelimitedText (494).

Declaration: `Property Delimiter : Char`

Visibility: public

Access: Read,Write

Description: `Delimiter` is the delimiter character used to separate the different strings in the stringlist when they are read or set through the DelimitedText (494) property.

See also: TStrings.DelimitedText (494)

4.73.56 TStrings.Encoding

Synopsis: Current encoding of stringlist.

Declaration: `Property Encoding : TEncoding`

Visibility: public

Access: Read

Description: `Encoding` is the current encoding used by the `TStrings` instance, and which was specified in the constructor. It is not the default encoding, as specified by DefaultEncoding (494).

See also: DefaultEncoding (494)

4.73.57 TStrings.LineBreak

Synopsis: LineBreak character to use.

Declaration: `Property LineBreak : string`

Visibility: public

Access: Read,Write

Description: `LineBreak` is the character used to separate lines when reading or writing TStrings.Text (499), and it is also used when reading from file.

See also: TStrings.Text (499)

4.73.58 TStrings.MissingNameValueSeparatorAction

Synopsis: What to do if `NameValueSeparator` is missing in a string.

Declaration: `Property MissingNameValueSeparatorAction : TMissingNameValueSeparatorAction`

Visibility: public

Access: Read,Write

Description: `MissingNameValueSeparatorAction` describes what to to if a given string in the list does not have a NameValueSeparator (496) character in it: consider the line a name, a value or an empty line, or raise an error. For a list of possible values and their explanation, see TMissingNameValueSeparatorAction (283)

See also: TMissingNameValueSeparatorAction (283), IndexOfName (272), Names (272), Values (272), GetNameValue (272), ValueFromIndex (272)

4.73.59 TStrings.Names

Synopsis: Name parts of the name-value pairs in the list.

Declaration: `Property Names[Index: Integer]: string`

Visibility: public

Access: Read

Description: `Names` provides indexed access to the names of the name-value pairs in the list. It returns the name part of the `Index`-th string in the list.

Remark The index is not an index based on the number of name-value pairs in the list. It is the name part of the name-value pair a string `Index` in the list. If the string at position `Index` is not a name-value pair (i.e. does not contain the equal sign (=)), then an empty name is returned.

See also: TStrings.Values (500), TStrings.IndexOfName (487)

4.73.60 TStrings.NameValueSeparator

Synopsis: Value of the character used to separate name,value pairs.

Declaration: `Property NameValueSeparator : Char`

Visibility: public

Access: Read,Write

Description: `NameValueSeparator` is the character used to separate name,value pair. By default, this is the equal sign (=), resulting in Name=Value pairs.

It can be set to a colon for Name : Value pairs.

4.73.61 TStrings.Objects

Synopsis: Indexed access to the objects associated with the strings in the list.

Declaration: `Property Objects[Index: Integer]: TObject`

Visibility: public

Access: Read,Write

Description: `Objects` provides indexed access to the objects associated to the strings in the list. `Index` is a zero-based index and must be in the range of `0` to `Count-1`.

Setting the `objects` property will not free the previously associated object, if there was one. The caller is responsible for freeing the object that was previously associated to the string.

`TStrings` does not implement any storage for objects. Reading the `Objects` property will always return `Nil`, Setting the property will have no effect. It is the responsibility of the descendant classes to provide storage for the associated objects.

Errors: If an `Index` outside the valid range is specified, an EStringListError (312) exception will be raised.

See also: TStrings.Strings (498), TStrings.IndexOfObject (487), TStrings.Names (496), TStrings.Values (500)

4.73.62 TStrings.Options

Synopsis: A set of TStringsOption (288) - various boolean properties.

Declaration: `Property Options : TStringsOptions`

Visibility: public

Access: Read,Write

Description: Set `Options` instead of the underlying boolean properties.

See also: TStringsOption (288)

4.73.63 TStrings.QuoteChar

Synopsis: Quote character used in DelimitedText (494).

Declaration: `Property QuoteChar : Char`

Visibility: public

Access: Read,Write

Description: `QuoteChar` is the character used by the DelimitedText (494) property to quote strings that have a space or non-printing character in it.

4.73.64 TStrings.SkipLastLineBreak

Synopsis: Do not add a linebreak to the last item.

Declaration: `Property SkipLastLineBreak : Boolean`

Visibility: public

Access: Read,Write

Description: `SkipLastLineBreak` can be set to `True` to omit a linebreak character after the last string in the TStrings.Text (499). This also means when writing to file, that the file will not have a terminating linebreak character.

Note that `SkipLastLineBreak` has the opposite meaning to TStrings.TrailingLineBreak (497).

See also: TStrings.TrailingLineBreak (497), TStrings.LineBreak (495), TStrings.Text (499), TStrings.Options (497)

4.73.65 TStrings.TrailingLineBreak

Synopsis: Add a linebreak to the last item.

Declaration: `Property TrailingLineBreak : Boolean`

Visibility: public

Access: Read,Write

Description: `TrailingLineBreak` can be set to `False` to omit a linebreak character after the last string in the TStrings.Text (499). This also means when writing to file, that the file will not have a terminating linebreak character.

Note that `TrailingLineBreak` has the opposite meaning to TStrings.SkipLastLineBreak (497).

See also: TStrings.SkipLastLineBreak (497), TStrings.LineBreak (495), TStrings.Text (499), TStrings.Options (497)

4.73.66 TStrings.StrictDelimiter

Synopsis: Should only the delimiter character be considered a delimiter.

Declaration: `Property StrictDelimiter : Boolean`

Visibility: public

Access: Read,Write

Description: `StrictDelimiter` can be used to indicate that only the delimiter character should be considered a delimiter when setting DelimitedText (494): under normal circumstances, quotes and spaces are considered specially (see the TStrings.CommaText (493) property for more information).

When `StrictDelimiter` is set to `True` then only the Delimiter (495) character is considered when splitting the text in items: no quoting whatsoever is performed when writing the `DelimitedText` property. However, when reading the `DelimitedText` property, quoted strings are taken into account (so a quoted string can contain a delimiter that is treated as text instead of a delimiter).

See also: DelimitedText (494), CommaText (493), Delimiter (495), TStrings.Options (497)

4.73.67 TStrings.Strings

Synopsis: Indexed access to the strings in the list.

Declaration: `Property Strings[Index: Integer]: string; default`

Visibility: public

Access: Read,Write

Description: `Strings` is the default property of `TStrings`. It provides indexed read-write access to the list of strings. Reading it will return the string at position `Index` in the list. Writing it will set the string at position `Index`.

`Index` is the position of the string in the list. It is zero-based, i.e. valued values range from `0` (the first string in the list) till `Count-1` (the last string in the list). When browsing through the strings in the list, this fact must be taken into account.

To access the objects associated with the strings in the list, use the TStrings.Objects (496) property. The name parts of name-value pairs can be accessed with the TStrings.Names (496) property, and the values can be set or read through the TStrings.Values (500) property.

Searching through the list can be done using the TStrings.IndexOf (486) method.

Errors: If `Index` is outside the allowed range, an EStringListError (312) exception is raised.

See also: TStrings.Count (493), TStrings.Objects (496), TStrings.Names (496), TStrings.Values (500), TStrings.IndexOf (486)

4.73.68 TStrings.StringsAdapter

Synopsis: Not implemented in Free Pascal.

Declaration: `Property StringsAdapter : IStringsAdapter`

Visibility: public

Access: Read,Write

Description: Not implemented in Free Pascal.

4.73.69 TStrings.Text

Synopsis: Contents of the list as one big string.

Declaration: `Property Text : string`

Visibility: public

Access: Read,Write

Description: `Text` returns, when read, the contents of the stringlist as one big string consisting of all strings in the list, separated by an end-of-line marker. When this property is set, the string will be cut into smaller strings, based on the positions of end-of-line markers in the string. Any previous content of the stringlist will be lost.

Remark If any of the strings in the list contains an end-of-line marker, then the resulting string will appear to contain more strings than actually present in the list. To avoid this ambiguity, use the TStrings.CommaText (493) property instead.

See also: TStrings.Strings (498), TStrings.Count (493), TStrings.CommaText (493)

4.73.70 TStrings.TextLineBreakStyle

Synopsis: Determines which line breaks to use in the Text (499) property.

Declaration: `Property TextLineBreakStyle : TTextLineBreakStyle`

Visibility: public

Access: Read,Write

Description: `TextLineBreakStyle` determines which linebreak style is used when constructing the Text property: the same rules are used as in the writing to text files:

tlbsLFLines are separated with a linefeed character #10.

tlbsCRLFLines are separated with a carriage-return/linefeed character pair: #13#10.

tlbsCRLines are separated with a carriage-return character #13.

It has no effect when setting the text property.

See also: Text (499)

4.73.71 TStrings.UseLocale

Synopsis: Determines what methods are used in strings comparison.

Declaration: `Property UseLocale : Boolean`

Visibility: public

Access: Read,Write

Description: Set `UseLocale` to `True` to use AnsiCompareStr (272) and AnsiCompareText (272) for comparing strings in the `TStrings` object.

Set `UseLocale` to `False` to use CompareStr (272) and CompareText (272) for comparing strings in the `TStrings` object.

The default value is `True`.

See also: TStrings.Options (497)

4.73.72 TStrings.ValueFromIndex

Synopsis: Return the value part of a string based on it's index.

Declaration: `Property ValueFromIndex[Index: Integer]: string`

Visibility: public

Access: Read,Write

Description: `ValueFromIndex` returns the value part of a string based on the string index. The value part are all characters in the string after the NameValueSeparator (496) character, or all characters if the `NameValueSeparator` character is not present.

4.73.73 TStrings.Values

Synopsis: Value parts of the name-value pairs in the list.

Declaration: `Property Values[Name: string]: string`

Visibility: public

Access: Read,Write

Description: `Values` represents the value parts of the name-value pairs in the list.

When reading this property, if there is a name-value pair in the list of strings that has name part `Name`, then the corresponding value is returned. If there is no such pair, an empty string is returned.

When writing this value, first it is checked whether there exists a name-value pair in the list with name `Name`. If such a pair is found, it's value part is overwritten with the specified value. If no such pair is found, a new name-value pair is added with the specified `Name` and value.

Remark

1. Names are compared case-insensitively.
2. Any character, including whitespace, up till the first equal (=) sign in a string is considered part of the name.

See also: TStrings.Names (496), TStrings.Strings (498), TStrings.Objects (496)

4.73.74 TStrings.WriteBOM

Synopsis: Write BOM when writing stringlist to stream.

Declaration: `Property WriteBOM : Boolean`

Visibility: public

Access: Read,Write

Description: `WriteBOM` signals the stringlist to write a BOM Marker to the stream if the stringlist is written to stream using TStrings.SaveToStream (491) or TStrings.SaveToFile (491)

See also: TStrings.SaveToStream (491), TStrings.SaveToFile (491), TStrings.Options (497)

4.74 TStringsEnumerator

4.74.1 Description

`TStringsEnumerator` implements the #rtl.system.IEnumerator (1617) interface for the TStrings (475) class, so the `TStrings` class can be used in a `for ... in` loop. It is returned by the TStrings.GetEnumerator (486) method of `TStrings`.

See also: TStrings (475), TStrings.GetEnumerator (486), #rtl.system.IEnumerator (1617)

4.74.2 Method overview

Page	Method	Description
501	Create	Initialize a new instance of `TStringsEnumerator`.
501	GetCurrent	Return the current pointer in the list.
501	MoveNext	Move the position of the enumerator to the next position in the list.

4.74.3 Property overview

Page	Properties	Access	Description
502	Current	r	Current pointer in the list.

4.74.4 TStringsEnumerator.Create

Synopsis: Initialize a new instance of `TStringsEnumerator`.

Declaration:
```
constructor Create(AStrings: TStrings)
```

Visibility: public

Description: `Create` initializes a new instance of `TStringsEnumerator` and keeps a reference to the stringlist `AStrings` that will be enumerated.

See also: TStrings (475)

4.74.5 TStringsEnumerator.GetCurrent

Synopsis: Return the current pointer in the list.

Declaration:
```
function GetCurrent : string
```

Visibility: public

Description: `GetCurrent` returns the current string item in the enumerator.

Errors: No checking is done on the validity of the current position.

See also: MoveNext (501), TStringItem (309)

4.74.6 TStringsEnumerator.MoveNext

Synopsis: Move the position of the enumerator to the next position in the list.

Declaration:
```
function MoveNext : Boolean
```

Visibility: public

Description: `MoveNext` puts the pointer on the next item in the stringlist, and returns `True` if this succeeded, or `False` if the pointer is past the last element in the list.

Errors: Note that if `False` is returned, calling `GetCurrent` will result in an exception.

See also: GetCurrent (501)

4.74.7 TStringsEnumerator.Current

Synopsis: Current pointer in the list.

Declaration: `Property Current : string`

Visibility: public

Access: Read

Description: `Current` redefines GetCurrent (501) as a property.

See also: GetCurrent (501)

4.75 TStringStream

4.75.1 Description

`TStringStream` stores its data in an ansistring. The contents of this string is available as the DataString (505) property. It also introduces some methods to read or write parts of the stringstream's data as a string.

The main purpose of a `TStringSTream` is to be able to treat a string as a stream from which can be read.

See also: TStream (455), TStringStream.DataString (505), TStringStream.ReadString (505), TStringStream.WriteString (505)

4.75.2 Method overview

Page	Method	Description
503	Create	Creates a new stringstream and sets its initial content.
503	CreateRaw	Create stringstream using codepage of string.
503	Destroy	Free the instance of the stream.
504	ReadAnsiString	Read an ansistring from the stream.
505	ReadString	Reads a string of length `Count`.
504	ReadUnicodeString	Read a unicode string from the stream.
504	WriteAnsiString	Write an ansistring to the stream.
505	WriteString	`WriteString` writes a string to the stream.
504	WriteUnicodeString	Write a unicode string to the stream.

4.75.3 Property overview

Page	Properties	Access	Description
505	DataString	r	Contains the contents of the stream in string form.
506	Encoding	r	Encoding of the string with the data.
506	OwnsEncoding	r	Does the stream instance owns the encoding.
505	UnicodeDataString	r	Datastring as unicode string.

4.75.4 TStringStream.Create

Synopsis: Creates a new stringstream and sets its initial content.

Declaration:
```
constructor Create(const ABytes: TBytes);  Override;  Overload
constructor Create(const AString: string);  Overload
constructor Create(const AString: string; AEncoding: TEncoding;
                   AOwnsEncoding: Boolean);  Overload
constructor Create(const AString: string; ACodePage: Integer);  Overload
constructor Create(const AString: UnicodeString);  Overload
constructor Create(const AString: UnicodeString; AEncoding: TEncoding;
                   AOwnsEncoding: Boolean);  Overload
constructor Create(const AString: UnicodeString; ACodePage: Integer)
                   ;  Overload
```

Visibility: public

Description: `Create` creates a new `TStringStream` instance and sets its initial content to `Astring`. The position is still 0 but the size of the stream will equal the length of the string.

The `Encoding` argument specifies the codepage with which the bytes in the string will be interpreted. If it is not specified (or Nil) the system default encoding will be used. The correct encoding can be detected from the string by using the TStringStream.CreateRaw (503) constructor.

See also: TStringStream.DataString (505), TStringStream.UnicodeDataString (505), TStringStream.Encoding (506), TStringStream.CreateRaw (503)

4.75.5 TStringStream.CreateRaw

Synopsis: Create stringstream using codepage of string.

Declaration:
```
constructor CreateRaw(const AString: RawByteString);  Overload
```

Visibility: public

Description: `CreateRaw` will create the stream using the codepage of the string passed on in the constructor. The default constructor uses the default codepage of the system.

See also: TStringStream.Create (503)

4.75.6 TStringStream.Destroy

Synopsis: Free the instance of the stream.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` frees the encoding if it owns it and calls the inherited destructor.

4.75.7 TStringStream.ReadUnicodeString

Synopsis: Read a unicode string from the stream.

Declaration: `function ReadUnicodeString(Count: LongInt) : UnicodeString`

Visibility: public

Description: `ReadUnicodeString` will read a unicodestring and correctly handle the codepage translation if necessary..

See also: TStream.ReadString (455), TStream.ReadAnsiString (463), TStringStream.WriteUnicodeString (504)

4.75.8 TStringStream.WriteUnicodeString

Synopsis: Write a unicode string to the stream.

Declaration: `procedure WriteUnicodeString(const AString: UnicodeString)`

Visibility: public

Description: `WriteUnicodeString` will write a unicodestring and correctly handle the codepage translation if necessary.

See also: TStream.ReadString (455), TStringStream.ReadUnicodeString (504), TStringStream.ReadAnsiString (504), TStringStream.WriteAnsiString (504)

4.75.9 TStringStream.ReadAnsiString

Synopsis: Read an ansistring from the stream.

Declaration: `function ReadAnsiString(Count: LongInt) : AnsiString;  Overload`

Visibility: public

Description: `ReadAnsiString` will read an ansistring and correctly handle the codepage translation if necessary..

See also: TStream.ReadString (455), TStringStream.WriteUnicodeString (504), TStringStream.WriteAnsiString (504), TStringStream.ReadUnicodeString (504)

4.75.10 TStringStream.WriteAnsiString

Synopsis: Write an ansistring to the stream.

Declaration: `procedure WriteAnsiString(const AString: AnsiString);  Override`

Visibility: public

Description: `WriteAnsiString` will write an ansistring and correctly handle the codepage translation if necessary.

See also: TStream.ReadString (455), TStringStream.ReadUnicodeString (504), TStringStream.ReadAnsiString (504), TStringStream.WriteUnicodeString (504)

4.75.11 TStringStream.ReadString

Synopsis: Reads a string of length `Count`.

Declaration: `function ReadString(Count: LongInt) : string`

Visibility: public

Description: `ReadString` reads `Count` bytes from the stream and returns the read bytes as a string. If less than `Count` bytes were available, the string has as many characters as bytes could be read.

The `ReadString` method is a wrapper around the Read (502) method. It does not do the same string as the TStream.ReadAnsiString (463) method, which first reads a length integer to determine the length of the string to be read.

See also: TStringStream.Read (502), TStream.ReadAnsiString (463)

4.75.12 TStringStream.WriteString

Synopsis: `WriteString` writes a string to the stream.

Declaration: `procedure WriteString(const AString: string)`

Visibility: public

Description: `WriteString` writes a string to the stream.

4.75.13 TStringStream.DataString

Synopsis: Contains the contents of the stream in string form.

Declaration: `Property DataString : string`

Visibility: public

Access: Read

Description: Contains the contents of the stream in string form.

4.75.14 TStringStream.UnicodeDataString

Synopsis: Datastring as unicode string.

Declaration: `Property UnicodeDataString : UnicodeString`

Visibility: public

Access: Read

Description: `UnicodeDataString` returns the data string as a unicode encoded string.

See also: TStringStream.DataString (505)

4.75.15 TStringStream.OwnsEncoding

Synopsis: Does the stream instance owns the encoding.

Declaration: `Property OwnsEncoding : Boolean`

Visibility: public

Access: Read

Description: `OwnsEncoding` indicates whether the stream instance owns the encoding or not. If it owns the encoding, it will be freed when the stream instance is freed.

See also: TStringStream.Encoding (506)

4.75.16 TStringStream.Encoding

Synopsis: Encoding of the string with the data.

Declaration: `Property Encoding : TEncoding`

Visibility: public

Access: Read

Description: `Encoding` is the encoding of the string which contains the data. If the encoding was not specified (or detected) in the constructor, this is the default system encoding.

See also: TStringStream.OwnsEncoding (506), TStringStream.Create (503), TStringStream.CreateRaw (503)

4.76 TTextObjectWriter

4.76.1 Description

Not yet implemented.

4.77 TThread

4.77.1 Description

The `TThread` class encapsulates the native thread support of the operating system. To create a thread, declare a descendant of the `TThread` object and override the Execute (507) method. In this method, the tthread's code should be executed. To run a thread, create an instance of the tthread descendant, and call it's execute method.

It is also possible to simply execute a method or static procedure in a thread using the TThread.ExecuteInThread (512) class method.

See also: EThread (312), TThread.Execute (507), TThread.ExecuteInThread (512)

4.77.2 Method overview

Page	Method	Description
517	AfterConstruction	Code to be executed after construction but before execute.
510	CheckTerminated	Check if the current thread has finished executing.
509	Create	Creates a new thread.
509	CreateAnonymousThread	Execute code in an anonymous thread.
509	Destroy	Destroys the thread object.
507	Execute	Execute method. Must be overridden in a descendant thread.
512	ExecuteInThread	Execute a method or static procedure in a thread.
508	ForceQueue	Always queue a method for execution in the main thread.
511	GetSystemTimes	Return CPU stats.
512	GetTickCount	Return tick count (32-bit).
512	GetTickCount64	Return tick count (64-bit).
509	NameThreadForDebugging	Set a thread name.
508	Queue	Queue a method for execution in the main thread.
510	RemoveQueuedEvents	Remove methods scheduled for execution from queue.
517	Resume	Resumes the thread's execution. Deprecated, see `TThread.Start`.
510	SetReturnValue	Set return value of a thread.
511	Sleep	Prevent thread execution.
511	SpinWait	Prevent thread execution in a spin-wait loop.
517	Start	Starts a thread that was created in a suspended state.
517	Suspend	Suspends the thread's execution.
508	Synchronize	Synchronizes the thread by executing the method in the main thread.
518	Terminate	Signals the thread it should terminate.
518	WaitFor	Waits for the thread to terminate and returns the exit status.
511	Yield	Yield execution to other threads.

4.77.3 Property overview

Page	Properties	Access	Description
518	CurrentThread	r	Return current thread instance.
520	ExternalThread	r	Is the thread instance an external thread ?
521	FatalException	r	Exception that occurred during thread execution.
521	Finished	r	Has the thread finished executing.
519	FreeOnTerminate	rw	Indicates whether the thread should free itself when it stops executing.
520	Handle	r	Returns the thread handle.
519	IsSingleProcessor	r	Is the current system single processor or not.
521	OnTerminate	rw	Event called when the thread terminates.
520	Priority	rw	Returns the thread priority.
519	ProcessorCount	r	Return the processor count for this system.
520	Suspended	rw	Indicates whether the thread is suspended.
521	ThreadID	r	Returns the thread ID.

4.77.4 TThread.Execute

Synopsis: Execute method. Must be overridden in a descendant thread.

Declaration: `procedure Execute; Virtual; Abstract`

Visibility: protected

Description: `Execute` is a method that must be overridden in descendant classes of the thread. It must contain the code that must execute in the thread. The `Execute` method is responsible for checking Terminated (272) at regular intervals: when it is set to `True` the execute method must exit.

See also: Terminated (272)

4.77.5 TThread.Synchronize

Synopsis: Synchronizes the thread by executing the method in the main thread.

Declaration:
```
procedure Synchronize(AMethod: TThreadMethod)
class procedure Synchronize(AThread: TThread; AMethod: TThreadMethod)
```

Visibility: protected

Description: Synchronizes the thread by executing the method in the main thread.

4.77.6 TThread.Queue

Synopsis: Queue a method for execution in the main thread.

Declaration:
```
procedure Queue(aMethod: TThreadMethod)
class procedure Queue(aThread: TThread; aMethod: TThreadMethod);  Static
```

Visibility: protected

Description: `Queue` schedules a method `aMethod` for execution in the main thread. In difference with TThread.Synchronize (508), `Queue` just posts the method for execution in a queue, and does not wait for it to be executed, so this call returns at once.

If the call is made in the main thread, the method is not put on the queue, but is executed at once.

In the class procedure overloaded version of this call, the thread for which the method must be posted is the first argument. In the protected version of this call (used in the tthread instance), this argument is not there, and the thread instance is used.

When a thread object is destroyed (after it has finished executing) all its queued calls are removed from the queue list.

See also: TThread.Synchronize (508), TThread.RemoveQueuedEvents (510), TTHread.ForceQueue (508)

4.77.7 TThread.ForceQueue

Synopsis: Always queue a method for execution in the main thread.

Declaration:
```
procedure ForceQueue(aMethod: TThreadMethod)
class procedure ForceQueue(aThread: TThread; aMethod: TThreadMethod)
                           ;  Static
```

Visibility: protected

Description: `ForceQueue` schedules a method `aMethod` for execution in the main thread, just like TThread.Queue (508). In difference with TThread.Queue (508), if the call is made from the main thread, the function will not be executed at once, bu will actuall be put in the queue.

See also: TThread.Synchronize (508), TThread.RemoveQueuedEvents (510), TThread.Queue (508)

4.77.8 TThread.Create

Synopsis: Creates a new thread.

Declaration:
```
constructor Create(CreateSuspended: Boolean; const StackSize: SizeUInt)
```

Visibility: public

Description: Creates a new thread.

4.77.9 TThread.Destroy

Synopsis: Destroys the thread object.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: Destroys the thread object.

4.77.10 TThread.CreateAnonymousThread

Synopsis: Execute code in an anonymous thread.

Declaration:
```
class function CreateAnonymousThread(aProc: TProcedure) : TThread
                                          ;  Static
```

Visibility: public

Description: `CreateAnonymousThread` will create an instance of a `TThread` descendant and calls `aProc` in this procedure. This can be used to quickly execute a method in another thread without having to explitiely declare a thread for such purposes. It returns the created `TThread` instance, which can be checked for termination etc.

Note that this method differs slightly from Delphi in that FPC does not yet support anonymous methods, so the signature of `aProc` differs slightly.

See also: TThread.CheckTerminated (510)

4.77.11 TThread.NameThreadForDebugging

Synopsis: Set a thread name.

Declaration:
```
class procedure NameThreadForDebugging(aThreadName: UnicodeString;
                                       aThreadID: TThreadID);  Static
class procedure NameThreadForDebugging(aThreadName: AnsiString;
                                       aThreadID: TThreadID);  Static
```

Visibility: public

Description: `NameThreadForDebugging` sets the name of thread `aThreadID` to `aThreadName`. The thread name can be Unicode or ansistring. This is mainly useful for debugging purposes, as thread names are more easily recognizable than IDs.

Note that this requires OS support, so at the moment it is Windows and Unix only.

4.77.12 TThread.SetReturnValue

Synopsis: Set return value of a thread.

Declaration:
```
class procedure SetReturnValue(aValue: Integer);  Static
```

Visibility: public

Description: `TThread.SetReturnValue` sets the return value of an internally created thread.

Errors: If the thread was not created by the FPC program, an EThreadExternalException (313) exception is raised.

See also: EThreadExternalException (313), TThread.CheckTerminated (510)

4.77.13 TThread.CheckTerminated

Synopsis: Check if the current thread has finished executing.

Declaration:
```
class function CheckTerminated : Boolean;  Static
```

Visibility: public

Description: `TThread.CheckTerminated` can be used to check if the current thread has finished executing (i.e. `Execute` has finished. This can be called from methods in other classes where the current thread instance is not available.

Errors: If the thread was not created by the FPC program, an EThreadExternalException (313) exception is raised.

See also: EThreadExternalException (313), TThread.SetReturnValue (510)

4.77.14 TThread.RemoveQueuedEvents

Synopsis: Remove methods scheduled for execution from queue.

Declaration:
```
class procedure RemoveQueuedEvents(aThread: TThread;
                                   aMethod: TThreadMethod);  Static
class procedure RemoveQueuedEvents(aMethod: TThreadMethod);  Static
class procedure RemoveQueuedEvents(aThread: TThread);  Static
```

Visibility: public

Description: `RemoveQueuedEvents` removes methods from the list of methods waiting for execution in the main thread.

If only `aThread` is specified, all methods scheduled for execution by that thread are removed.

If only `aMethod` is specified, then all calls to that method are removed, regardless of the thread. The methods are compared with both the Code and Data pointers (method code and method object).

If both arguments are specified, then all calls to the given method by the given thread are removed. This is a known difference to Delphi that deletes all given methods or all methods by the given thread.

See also: TThread.Synchronize (508), TThread.Queue (508)

4.77.15 TThread.SpinWait

Synopsis: Prevent thread execution in a spin-wait loop.

Declaration: `class procedure SpinWait(aIterations: LongWord); Static`

Visibility: public

Description: `SpinWait` blocks the execution of the thread in a spin-wait loop: it simply executes some simple instructions.

This can be used to create short time delays without an immediate thread switch (e.g. SysUtils.Sleep (272) can cause a thread switch). The input parameter (aIterations) specifies the number of spin loops.

See also: SysUtils.Sleep (272), TThread.Sleep (511)

4.77.16 TThread.Sleep

Synopsis: Prevent thread execution.

Declaration: `class procedure Sleep(aMilliseconds: Cardinal); Static`

Visibility: public

Description: `Sleep` blocks the execution of the thread for `aMilliseconds`. This function simply calls sysutils.sleep (272)

In difference with TThread.SpinWait (511), a thread switch may occur during the sleep.

See also: SysUtils.Sleep (272), TThread.SpinWait (511)

4.77.17 TThread.Yield

Synopsis: Yield execution to other threads.

Declaration: `class procedure Yield; Static`

Visibility: public

Description: `TThread.Yield` yields the processor to other threads. It can be called from methods outside the thread class itself.

4.77.18 TThread.GetSystemTimes

Synopsis: Return CPU stats.

Declaration: `class procedure GetSystemTimes(out aSystemTimes: TSystemTimes); Static`

Visibility: public

Description: `GetSystemTimes` is provided for Delphi compatibility only, it currently returns empty values only.

See also: TSystemTimes (272)

4.77.19 TThread.GetTickCount

Synopsis: Return tick count (32-bit).

Declaration:
```
class function GetTickCount : LongWord;  Static
```

Visibility: public

Description: `GetTickCount` is deprecated and should not be used. Use TThread.GetTickCount64 (512) instead.

See also: TThread.GetTickCount64 (512)

4.77.20 TThread.GetTickCount64

Synopsis: Return tick count (64-bit).

Declaration:
```
class function GetTickCount64 : QWord;  Static
```

Visibility: public

Description: `GetTickCount64` simply calls SysUtils.GetTickCount64 (272) and is implemented for Delphi compatibility only.

See also: SysUtils.GetTickCount64 (272)

4.77.21 TThread.ExecuteInThread

Synopsis: Execute a method or static procedure in a thread.

Declaration:
```
class function ExecuteInThread(AMethod: TThreadExecuteHandler;
                               AOnTerminate: TNotifyEvent) : TThread
                               ;  Overload;  Static
class function ExecuteInThread(AMethod: TThreadExecuteStatusHandler;
                               AOnStatus: TThreadStatusNotifyEvent;
                               AOnTerminate: TNotifyEvent) : TThread
                               ;  Overload;  Static
class function ExecuteInThread(AMethod: TThreadExecuteCallBack;
                               AData: Pointer;
                               AOnTerminate: TNotifyCallBack) : TThread
                               ;  Overload;  Static
class function ExecuteInThread(AMethod: TThreadExecuteStatusCallBack;
                               AOnStatus: TThreadStatusNotifyCallBack;
                               AData: Pointer;
                               AOnTerminate: TNotifyCallBack) : TThread
                               ;  Overload;  Static
```

Visibility: public

Description: `TThread.ExecuteInThread` is a class method which allows to quickly execute a method or procedure in a thread. The method or procedure to be executed is passed in `Method`, this can be a method or a plain (static) procedure.

The caller can be notified of thread termination: In the optional argument `AOnTerminate` a callback (procedure or method, depending on the signature) can be specified that will be called when the thread terminated. This callback is executed in the main thread.

The signature of `AMethod` determines whether status reporting is enabled or not. If the method of type TThreadExecuteStatusHandler (290) or TThreadExecuteStatusCallback (290), then an extra

AOnStatus callback must be specified. This callback will be called in the main thread whenever the thread wishes to be report its status. The status callback should not do extensive work, because while the status callback is called, thread execution is suspended.

When using a plain procedure, extra data can be passed on to the procedure in AData. The AData pointer will be passed to the thread method, and also to the thread status callback and thread termination callback.

See also: TThreadExecuteHandler (289), TThreadExecuteStatusHandler (290), TThreadExecuteStatusCallback (290), TThreadExecuteCallback (289)

Listing: ./examples/classesex/tthrc.pp

```
program tthrc;

uses cthreads, sysutils, classes;

Var
  D : Integer;

Procedure DoneThread(Sender : TObject; AData : Pointer);

begin
  Writeln('Thread ',TThread(Sender).ThreadID,' done. D is currently: ', PInteger(AData)^);
end;

Procedure DoThread(AData : Pointer);

Var
  I : integer;

begin
  for I:=1 to 10 do
    begin
    Sleep(10*Random(30));
    Writeln('Thread ',TThread.CurrentThread.ThreadID,' ping ',I);
    Inc(PInteger(AData)^,i);
    end;
end;

Var
  T1,T2 : TThread;

begin
  T1:=TThread.ExecuteInThread(@DoThread,@D,@DoneThread);
  T2:=TThread.ExecuteInThread(@DoThread,@D,@DoneThread);
  Writeln('Main thread done');
  T1.WaitFor;
  T2.WaitFor;
end.
```

Listing: ./examples/classesex/tthre.pp

```
program tthre;

{$mode objfpc}
{$H+}

uses cthreads, sysutils, classes;
```

```
Type
  TTestThread = Class(TObject)
    D : Integer;
    Procedure DoneThread(Sender : TObject);
    Procedure DoThread;
    Procedure Run;
  end;



Procedure TTestThread.DoneThread(Sender : TObject);

begin
  Writeln('Thread ',TThread(Sender).ThreadID,' done. D is currently: ', D);
end;

Procedure TTestThread.DoThread;

Var
  I : integer;

begin
  for I:=1 to 10 do
    begin
    Sleep(10*Random(30));
    Writeln('Thread ',TThread.CurrentThread.ThreadID,' ping ',I);
    Inc(D,i);
    end;
end;

Procedure TTestThread.Run;


Var
  T1,T2 : TThread;

begin
  T1:=TThread.ExecuteInThread(@DoThread,@DoneThread);
  T2:=TThread.ExecuteInThread(@DoThread,@DoneThread);
  Writeln('Main thread done');
  T1.WaitFor;
  T2.WaitFor;
end;

begin
  With TTestThread.Create do
    try
      Run;
    finally
      Free;
    end;
end.
```

Listing: ./examples/classesex/tthrcs.pp

```
program tthrcs;
{$h+}
```

```
uses cthreads, sysutils, classes;

Var
  D : Integer;
  DoneThreads : INteger;

Procedure DoneThread(Sender : TObject; Data : Pointer);

begin
  Inc(DoneThreads);
  Writeln('Thread ',TThread(Sender).ThreadID,' done. D is currently: ', PInteger(Data)^);
end;

Procedure ReportThreadStatus(Sender : TThread; AData : Pointer;Const status : String);

begin
   Writeln('Thread ',Sender.ThreadID,' Status report : ',Status);
end;


Procedure DoThread(AData : Pointer; Report : TThreadReportStatus);

Var
  I : integer;

begin
  for I:=1 to 10 do
    begin
    Sleep(10*Random(30));
    Report('Ping '+IntToStr(i));
    Inc(PInteger(AData)^,i);
    end;
end;

Var
  T1,T2 : TThread;

begin
  DoneThreads:=0;
  T1:=TThread.ExecuteInThread(@DoThread,@ReportThreadStatus,@D,@DoneThread);
  T2:=TThread.ExecuteInThread(@DoThread,@ReportThreadStatus,@D,@DoneThread);
  Writeln('Main thread loop');
  While DoneThreads<2 do
    begin
    Sleep(10);
    CheckSynchronize;
    end;
  T1.WaitFor;
  T2.WaitFor;
end.
```

Listing: ./examples/classesex/tthres.pp

```
program tthrc;

{$mode objfpc}
{$H+}

uses cthreads, sysutils, classes;
```

```
Type
  TTestThread = Class(TObject)
    D : Integer;
    DoneThreads : integer;
    Procedure DoneThread(Sender : TObject);
    Procedure ReportThreadStatus(Sender : TThread; Const status : String);
    Procedure DoThread(Report: TThreadReportStatus);
    Procedure Run;
  end;


Procedure TTestThread.DoneThread(Sender : TObject);

begin
  Inc(DoneThreads);
  Writeln('Thread ',TThread(Sender).ThreadID,' done. D is currently: ', D);
end;

Procedure TTestThread.ReportThreadStatus(Sender : TThread; Const status : String);

begin
   Writeln('Thread ',Sender.ThreadID,' Status report : ',Status);
end;

Procedure TTestThread.DoThread(Report : TThreadReportStatus);

Var
  I : integer;

begin
  for I:=1 to 10 do
    begin
    Sleep(10*Random(30));
    Report('Ping '+IntToStr(i));
    Inc(D,i);
    end;
end;

Procedure TTestThread.Run;


Var
  T1,T2 : TThread;

begin
  DoneThreads:=0;
  T1:=TThread.ExecuteInThread(@DoThread,@ReportThreadStatus,@DoneThread);
  T2:=TThread.ExecuteInThread(@DoThread,@ReportThreadStatus,@DoneThread);
  Writeln('Main thread loop');
  While DoneThreads<2 do
    begin
    Sleep(10);
    CheckSynchronize;
    end;
  T1.WaitFor;
  T2.WaitFor;
end;
```

```
begin
  With TTestThread.Create do
    try
      Run;
    finally
      Free;
    end;
end.
```

4.77.22 TThread.AfterConstruction

Synopsis: Code to be executed after construction but before execute.

Declaration: `procedure AfterConstruction; Override`

Visibility: public

Description: `AfterConstruction` is overridden in `TThread`, it actually starts the thread if it was created with `CreateSuspended` equal to `False`, i.e. not suspended. When overriding this method, the inherited method must be called, or the thread will never be started.

4.77.23 TThread.Start

Synopsis: Starts a thread that was created in a suspended state.

Declaration: `procedure Start`

Visibility: public

Description: The effect of this method is currently the same as calling `TThread.Resume` after creating a thread in a suspended state. This method was added for Delphi-compatibility, where it was introduced after `TThread.Suspend` and `TThread.Resume` were deprecated.

See also: TThread.Create (509)

4.77.24 TThread.Resume

Synopsis: Resumes the thread's execution. Deprecated, see `TThread.Start`.

Declaration: `procedure Resume`

Visibility: public

Description: Resumes the thread's execution. Deprecated, see `TThread.Start`.

See also: TThread.Start (517), TThread.Suspend (517)

4.77.25 TThread.Suspend

Synopsis: Suspends the thread's execution.

Declaration: `procedure Suspend`

Visibility: public

Description: On non-Windows platforms, a thread can only suspend itself. Other threads can wake up a suspended thread by calling `TThread.Start`.

See also: TThread.Resume (517), TThread.Start (517)

4.77.26 TThread.Terminate

Synopsis: Signals the thread it should terminate.

Declaration: `procedure Terminate`

Visibility: public

Description: `Terminate` sets the TThread.Terminated (506) flag. It does not in any way attempt to terminate the thread in any other way, this just signals the thread that it should stop executing at the earliest possible moment.

See also: TThread.Terminated (506), TThread.WaitFor (518), TThread.FreeOnTerminate (519), OnTerminate (521)

4.77.27 TThread.WaitFor

Synopsis: Waits for the thread to terminate and returns the exit status.

Declaration: `function WaitFor : Integer`

Visibility: public

Description: `WaitFor` waits for the thread to terminate, and returns the exit status. Note that when executed in the main thread, this method calls CheckSynchronize (295), this is done to avoid deadlocks: if the thread is waiting for a synchronize (508), then the synchronize methods will be executed and then the `WaitFor` will return.

See also: TThread.Terminated (506), TThread.WaitFor (518), TThread.FreeOnTerminate (519), TThread.Synchronize (508)

4.77.28 TThread.CurrentThread

Synopsis: Return current thread instance.

Declaration: `Property CurrentThread : TThread`

Visibility: public

Access: Read

Description: `TThread.CurrentThread` can be used to get the current thread instance. This is useful in code that is not inside a TThread implementation, but which needs access to the current thread.

For threads that were created outside of FPC code (DLLs or a calling program) this will return a dummy TThread instance.

See also: TThread.ExternalThread (520)

4.77.29 TThread.ProcessorCount

Synopsis: Return the processor count for this system.

Declaration: `Property ProcessorCount : LongWord`

Visibility: public

Access: Read

Description: `ProcessorCount` returns the processor count for this system.

Whether this is the number of cores or the number of CPUs present in the hardware, is deliberately unspecified. The number of cores can also vary during the lifetime of the program, and the FPC implementation does not guarantee that this will always match, the value is set at program start.

As such, the number specified should only be used as an indication of how many threads can be executed at once by the system.

See also: TThread.IsSingleProcessor (519)

4.77.30 TThread.IsSingleProcessor

Synopsis: Is the current system single processor or not.

Declaration: `Property IsSingleProcessor : Boolean`

Visibility: public

Access: Read

Description: `Thread.IsSingleProcessor` returns `True` if TThread.ProcessorCount (519) is less than or equal to 1, `False` otherwise.

See also: TThread.ProcessorCount (519)

4.77.31 TThread.FreeOnTerminate

Synopsis: Indicates whether the thread should free itself when it stops executing.

Declaration: `Property FreeOnTerminate : Boolean`

Visibility: public

Access: Read,Write

Description: `FreeOnTerminate`, when set to `True` indicates that the tread instance will freed automatically as soon as the thread stops executing. You can use the OnTerminate (521) property to get a notification of when the thread has terminated and will be freed.

When setting this property to `True`, in general you may not read or write any property of the `TThread` instance from a different thread, because there is no guarantee that the thread instance still exists in memory. This implies 2 things:

1. The `OnTerminate` event handler should be set before setting `FreeOnTerminate` to `True`
2. The properties can still be read and set in the `OnTerminate` event handler, as the thread instance is then still guaranteed to exist.

If `FreeOnTerminate` is set to `False`, to stop and delete a running thread from another thread, the following sample code can be used:

```
aThread.Terminate;
aThread.WaitFor;
FreeAndNil(aThread);
```

See also: OnTerminate (521)

4.77.32 TThread.Handle

Synopsis: Returns the thread handle.

Declaration: `Property Handle : TThreadID`

Visibility: public

Access: Read

Description: Returns the thread handle.

4.77.33 TThread.ExternalThread

Synopsis: Is the thread instance an external thread ?

Declaration: `Property ExternalThread : Boolean`

Visibility: public

Access: Read

Description: `ExternalThread` returns `True` if the thread is an externally created thread. If the thread was created by the FPC program, this returns `False`. This is useful for examining instances returned by TThread.CurrentThread (518).

See also: TThread.CurrentThread (518)

4.77.34 TThread.Priority

Synopsis: Returns the thread priority.

Declaration: `Property Priority : TThreadPriority`

Visibility: public

Access: Read,Write

Description: Returns the thread priority.

4.77.35 TThread.Suspended

Synopsis: Indicates whether the thread is suspended.

Declaration: `Property Suspended : Boolean`

Visibility: public

Access: Read,Write

Description: Indicates whether the thread is suspended.

4.77.36 TThread.Finished

Synopsis: Has the thread finished executing.

Declaration: `Property Finished : Boolean`

Visibility: public

Access: Read

Description: `Finished` is `True` when TThread.Execute (507) has finished executing, but the thread is still cleaning up (calling OnTerminate, etc).

See also: TThread.Execute (507), TThread.OnTerminate (521)

4.77.37 TThread.ThreadID

Synopsis: Returns the thread ID.

Declaration: `Property ThreadID : TThreadID`

Visibility: public

Access: Read

Description: Returns the thread ID.

4.77.38 TThread.OnTerminate

Synopsis: Event called when the thread terminates.

Declaration: `Property OnTerminate : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `TThread.OnTerminate` is called when the thread terminates. The event is always called in the context of the main thread, i.e. using TThread.Synchronize (508)

See also: TThread.Synchronize (508), TThread.FreeOnTerminate (519)

4.77.39 TThread.FatalException

Synopsis: Exception that occurred during thread execution.

Declaration: `Property FatalException : TObject`

Visibility: public

Access: Read

Description: `FatalException` contains the exception that occurred during the thread's execution.

4.78 TThreadList

4.78.1 Description

TThreadList is a thread-safe Tlist (415) implementation. Unlike `TList`, it can be accessed read-write by multiple threads: the list implementation will take care of locking the list when adding or removing items from the list.

See also: TList (415)

4.78.2 Method overview

Page	Method	Description
523	Add	Adds an element to the list.
523	Clear	Removes all elements from the list.
522	Create	Creates a new thread-safe list.
522	Destroy	Destroys the list instance.
523	LockList	Locks the list for exclusive access.
523	Remove	Removes an item from the list.
524	UnlockList	Unlocks the list after it was locked.

4.78.3 Property overview

Page	Properties	Access	Description
524	Duplicates	rw	Describes what to do with duplicates.

4.78.4 TThreadList.Create

Synopsis: Creates a new thread-safe list.

Declaration: `constructor Create`

Visibility: public

Description: `Create` instantiates a new `TThreadList` instance. It initializes a critical section and an internal list object.

See also: TThreadList.Destroy (522)

4.78.5 TThreadList.Destroy

Synopsis: Destroys the list instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` finalizes the critical section, clears the internal list object and calls the inherited destructor.

See also: TThreadList.Create (522)

4.78.6 TThreadList.Add

Synopsis: Adds an element to the list.

Declaration: `procedure Add(Item: Pointer)`

Visibility: public

Description: `Add` attempts to lock the list and adds the pointer `Item` to the list. After the pointer was added, the list is unlocked again.

See also: LockList (523), Clear (523), Remove (523), UnlockList (524)

4.78.7 TThreadList.Clear

Synopsis: Removes all elements from the list.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` attempts to lock the list and then clears the list; all items are removed from the list. After the list is cleared, it is again unlocked.

See also: LockList (523), Add (523), Remove (523), UnlockList (524)

4.78.8 TThreadList.LockList

Synopsis: Locks the list for exclusive access.

Declaration: `function LockList : TList`

Visibility: public

Description: `LockList` locks the thread list for exclusive access. The return value for the method is the internal TList instance with pointers to the TThread instances for the thread list. Locklist uses an internal critical section, so all rules for multiple locking of critical sections apply to locklist/unlocklist as well.

See also: Clear (523), Add (523), Remove (523), UnlockList (524)

4.78.9 TThreadList.Remove

Synopsis: Removes an item from the list.

Declaration: `procedure Remove(Item: Pointer)`

Visibility: public

Description: `Remove` attempts to lock the list and then removes `Item` from the list. After the item is removed, the list is again unlocked.

See also: LockList (523), Add (523), Clear (523), UnlockList (524)

4.78.10 TThreadList.UnlockList

Synopsis: Unlocks the list after it was locked.

Declaration: `procedure UnlockList`

Visibility: public

Description: `UnLockList` unlocks the list when it was locked for exclusive access. UnLocklist and LockList use an internal critical section, so all rules for multiple locking/unlocking of critical sections apply.

See also: Clear (523), Add (523), Remove (523), lockList (523)

4.78.11 TThreadList.Duplicates

Synopsis: Describes what to do with duplicates.

Declaration: `Property Duplicates : TDuplicates`

Visibility: public

Access: Read,Write

Description: `Duplicates` describes what the threadlist should do when a duplicate pointer is added to the list. It is identical in behaviour to the Duplicates (473) property of TStringList (470).

See also: TDuplicates (279)

4.79 TWriter

4.79.1 Description

The `TWriter` class is a writer class that implements generic component streaming capabilities, independent of the format of the data in the stream. It uses a driver class TAbstractObjectWriter (333) to do the actual reading of data. The interface of the `TWriter` class should be identical to the interface in Delphi.

Note that the `TWriter` design is such that it will write a single component to a stream. It will write all children of this component, but it is not designed to write multiple components in succession to one stream.

It should never be necessary to create an instance of this class directly. Instead, the TStream.WriteComponent (460) call should be used.

See also: TFiler (393), TWriter (524), TAbstractObjectReader (325)

4.79.2 Method overview

Page	Method	Description
525	Create	Creates a new Writer with a stream and bufsize.
526	DefineBinaryProperty	Callback used when defining and streaming custom properties.
526	DefineProperty	Callback used when defining and streaming custom properties.
526	Destroy	Destroys the writer instance.
526	FlushBuffer	Flush the buffer
526	Write	Write raw data to stream.
527	WriteBoolean	Write boolean value to the stream.
527	WriteChar	Write a character to the stream.
527	WriteCollection	Write a collection to the stream.
527	WriteComponent	Stream a component to the stream.
528	WriteCurrency	Write a currency value to the stream.
528	WriteDate	Write a date to the stream.
528	WriteDescendent	Write descendent to stream.
528	WriteFloat	Write a float to the stream.
529	WriteIdent	Write an identifier to the stream.
529	WriteInteger	Write an integer to the stream.
529	WriteListBegin	Write a start-of-list marker to the stream.
529	WriteListEnd	Write an end-of-list marker to the stream.
530	WriteRootComponent	Write a root component to the stream.
529	WriteSet	Write a set value to the stream.
530	WriteSignature	Write a signature to the stream.
528	WriteSingle	Write a single-type real to the stream.
530	WriteString	Write a string to the stream.
530	WriteUnicodeString	Write a Unicode string to the stream.
531	WriteVariant	Write a variant to the stream.
527	WriteWideChar	Write widechar to stream.
530	WriteWideString	Write a widestring value to the stream.

4.79.3 Property overview

Page	Properties	Access	Description
532	Driver	r	Driver used when writing to the stream.
531	OnFindAncestor	rw	Event occurring when an ancestor component must be found.
531	OnWriteMethodProperty	rw	Handler from writing method properties.
532	OnWriteStringProperty	rw	Event handler for translating strings written to stream.
532	PropertyPath	r	Path to the property that is currently being written.
531	RootAncestor	rw	Ancestor of root component.

4.79.4 TWriter.Create

Synopsis: Creates a new Writer with a stream and bufsize.

Declaration:
```
constructor Create(ADriver: TAbstractObjectWriter)
constructor Create(Stream: TStream; BufSize: Integer)
```

Visibility: public

Description: Creates a new Writer with a stream and bufsize.

4.79.5 TWriter.Destroy

Synopsis: Destroys the writer instance.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: Destroys the writer instance.

4.79.6 TWriter.FlushBuffer

Synopsis: Flush the buffer

Declaration:
```
procedure FlushBuffer;  Override
```

Visibility: public

Description: `FlushBuffer` flushes the buffer. It is provided for Delphi compatibility, and is not used in FPC.

See also: TFiler.FlushBuffer (394), TBinaryObjectWriter.FlushBuffer (354), TAbstractObjectWriter.FlushBuffer (335)

4.79.7 TWriter.DefineProperty

Synopsis: Callback used when defining and streaming custom properties.

Declaration:
```
procedure DefineProperty(const Name: string; ReadData: TReaderProc;
                         AWriteData: TWriterProc; HasData: Boolean)
                         ;  Override
```

Visibility: public

Description: Callback used when defining and streaming custom properties.

4.79.8 TWriter.DefineBinaryProperty

Synopsis: Callback used when defining and streaming custom properties.

Declaration:
```
procedure DefineBinaryProperty(const Name: string;
                               ReadData: TStreamProc;
                               AWriteData: TStreamProc; HasData: Boolean)
                               ;  Override
```

Visibility: public

Description: Callback used when defining and streaming custom properties.

4.79.9 TWriter.Write

Synopsis: Write raw data to stream.

Declaration:
```
procedure Write(const Buffer; Count: LongInt);  Virtual
```

Visibility: public

Description: `Write` is introduced for Delphi compatibility to write raw data to the component stream. This should not be used in new production code as it will totally mess up the streaming.

See also: TBinaryObjectWriter.Write (355), TAbstractObjectWriter.Write (336)

4.79.10 TWriter.WriteBoolean

Synopsis: Write boolean value to the stream.

Declaration: `procedure WriteBoolean(Value: Boolean)`

Visibility: public

Description: Write boolean value to the stream.

4.79.11 TWriter.WriteCollection

Synopsis: Write a collection to the stream.

Declaration: `procedure WriteCollection(Value: TCollection)`

Visibility: public

Description: Write a collection to the stream.

4.79.12 TWriter.WriteComponent

Synopsis: Stream a component to the stream.

Declaration: `procedure WriteComponent(Component: TComponent)`

Visibility: public

Description: Stream a component to the stream.

4.79.13 TWriter.WriteChar

Synopsis: Write a character to the stream.

Declaration: `procedure WriteChar(Value: Char)`

Visibility: public

Description: Write a character to the stream.

4.79.14 TWriter.WriteWideChar

Synopsis: Write widechar to stream.

Declaration: `procedure WriteWideChar(Value: WideChar)`

Visibility: public

Description: `WriteWideChar` writes a widechar to the stream. This actually writes a widestring of length 1.

See also: TReader.ReadWideChar (445), TWriter.WriteWideString (530)

4.79.15 TWriter.WriteDescendent

Synopsis: Write descendent to stream.

Declaration: `procedure WriteDescendent(ARoot: TComponent; AAncestor: TComponent)`

Visibility: public

Description: `WriteDescendent` writes `ARoot` as a descendent of `AAncestor` This is used to create diff streams: only the properties where `ARoot` differs from `AAncestor` are written to the stream.

See also: TStream.WriteDescendent (460)

4.79.16 TWriter.WriteFloat

Synopsis: Write a float to the stream.

Declaration: `procedure WriteFloat(const Value: Extended)`

Visibility: public

Description: Write a float to the stream.

4.79.17 TWriter.WriteSingle

Synopsis: Write a single-type real to the stream.

Declaration: `procedure WriteSingle(const Value: Single)`

Visibility: public

Description: Write a single-type real to the stream.

4.79.18 TWriter.WriteDate

Synopsis: Write a date to the stream.

Declaration: `procedure WriteDate(const Value: TDateTime)`

Visibility: public

Description: Write a date to the stream.

4.79.19 TWriter.WriteCurrency

Synopsis: Write a currency value to the stream.

Declaration: `procedure WriteCurrency(const Value: Currency)`

Visibility: public

Description: `WriteCurrency` writes a currency typed value to the stream. This method does nothing except call the driver method of the driver being used.

See also: TReader.ReadCurrency (446)

4.79.20 TWriter.WriteIdent

Synopsis: Write an identifier to the stream.

Declaration:
```
procedure WriteIdent(const Ident: string)
```

Visibility: public

Description: Write an identifier to the stream.

4.79.21 TWriter.WriteInteger

Synopsis: Write an integer to the stream.

Declaration:
```
procedure WriteInteger(Value: LongInt);  Overload
procedure WriteInteger(Value: Int64);  Overload
```

Visibility: public

Description: Write an integer to the stream.

4.79.22 TWriter.WriteSet

Synopsis: Write a set value to the stream.

Declaration:
```
procedure WriteSet(Value: LongInt; SetType: Pointer)
```

Visibility: public

Description: `WriteSet` writes a set `Value` consisting of elements with type `EnumType`. The set must be encoded as an integer where each element is encoded in a bit of the integer. Thus, at most an enumerated type with 32 elements can be written with this method.

Errors: No checking is performed on the validity of `EnumType`. It is assumed to be a valid `PTypeInfo` pointer.

See also: TReader.ReadSet (447)

4.79.23 TWriter.WriteListBegin

Synopsis: Write a start-of-list marker to the stream.

Declaration:
```
procedure WriteListBegin
```

Visibility: public

Description: Write a start-of-list marker to the stream.

4.79.24 TWriter.WriteListEnd

Synopsis: Write an end-of-list marker to the stream.

Declaration:
```
procedure WriteListEnd
```

Visibility: public

Description: Write an end-of-list marker to the stream.

4.79.25 TWriter.WriteSignature

Synopsis: Write a signature to the stream.

Declaration: `procedure WriteSignature`

Visibility: public

Description: `WriteSignature` writes the streaming signature (if any) to a stream. It is called once, at the start of writing the root component to a stream.

See also: TAbstractObjectWriter.WriteSignature (334), TBinaryObjectWriter.WriteSignature (353)

4.79.26 TWriter.WriteRootComponent

Synopsis: Write a root component to the stream.

Declaration: `procedure WriteRootComponent(ARoot: TComponent)`

Visibility: public

Description: Write a root component to the stream.

4.79.27 TWriter.WriteString

Synopsis: Write a string to the stream.

Declaration: `procedure WriteString(const Value: string)`

Visibility: public

Description: Write a string to the stream.

4.79.28 TWriter.WriteWideString

Synopsis: Write a widestring value to the stream.

Declaration: `procedure WriteWideString(const Value: WideString)`

Visibility: public

Description: `WriteWidestring` writes a currency typed value to the stream. This method does nothing except call the driver method of the driver being used.

See also: TReader.ReadWideString (449)

4.79.29 TWriter.WriteUnicodeString

Synopsis: Write a Unicode string to the stream.

Declaration: `procedure WriteUnicodeString(const Value: UnicodeString)`

Visibility: public

Description: `WriteUnicodeString` writes `Value`, a UnicodeString string to the stream. It simply passes the string on to the `WriteUnicodeString` method of the writer driver class.

See also: TBinaryObjectWriter.WriteUnicodeString (357), TReader.ReadUnicodeString (449)

4.79.30 TWriter.WriteVariant

Synopsis: Write a variant to the stream.

Declaration: `procedure WriteVariant(const VarValue: Variant)`

Visibility: public

Description: `WriteVariant` writes `Value`, a simple variant, o the stream. It simply passes the string on to the `WriteVariant` method of the writer driver class.

See also: TBinaryObjectWriter.WriteVariant (358), TReader.ReadVariant (448)

4.79.31 TWriter.RootAncestor

Synopsis: Ancestor of root component.

Declaration: `Property RootAncestor : TComponent`

Visibility: public

Access: Read,Write

Description: Ancestor of root component.

4.79.32 TWriter.OnFindAncestor

Synopsis: Event occurring when an ancestor component must be found.

Declaration: `Property OnFindAncestor : TFindAncestorEvent`

Visibility: public

Access: Read,Write

Description: Event occurring when an ancestor component must be found.

4.79.33 TWriter.OnWriteMethodProperty

Synopsis: Handler from writing method properties.

Declaration: `Property OnWriteMethodProperty : TWriteMethodPropertyEvent`

Visibility: public

Access: Read,Write

Description: `OnWriteMethodProperty` can be set by an IDE or some streaming mechanism which handles dummy values for method properties; It can be used to write a real value to the stream which will be interpreted correctly when the stream is read. See TWriteMethodPropertyEvent (293) for a description of the arguments.

See also: TWriteMethodPropertyEvent (293), TReader.OnSetMethodProperty (451)

4.79.34 TWriter.OnWriteStringProperty

Synopsis: Event handler for translating strings written to stream.

Declaration: `Property OnWriteStringProperty : TReadWriteStringPropertyEvent`

Visibility: public

Access: Read,Write

Description: `OnWriteStringProperty` is called whenever a string property is written to the stream. It can be used e.g. by a translation mechanism to translate the strings on the fly, when a form is written. See TReadWriteStringPropertyEvent (285) for a description of the various parameters.

See also: TReader.OnPropertyNotFound (450), TReader.OnSetMethodProperty (451), TReadWriteStringPropertyEvent (285)

4.79.35 TWriter.Driver

Synopsis: Driver used when writing to the stream.

Declaration: `Property Driver : TAbstractObjectWriter`

Visibility: public

Access: Read

Description: Driver used when writing to the stream.

4.79.36 TWriter.PropertyPath

Synopsis: Path to the property that is currently being written.

Declaration: `Property PropertyPath : string`

Visibility: public

Access: Read

Description: `PropertyPath` is set to the property name of the class currently being written to stream. This is only done when TPersistent (435) descendant class properties are written.

Chapter 5

Reference for unit 'clocale'

5.1 Used units

Table 5.1: Used units by unit 'clocale'

Name	Page
System	1362

5.2 Overview

The `clocale` offers no API by itself: it just initializes the internationalization settings of the sysutils (1635) unit with the values provided by the C library found on most Unix or Linux systems that are POSIX compliant.

The clocale should simply be included in the uses clause of the program, preferably as one of the first units, and the initialization section of the unit will do all the work.

Note that including this unit, links your program to the C library of the system.

It makes no sense to use this unit on a non-POSIX system: Windows, OS/2 or DOS - therefore it should always be between an ifdef statement:

```
program myprogram;

uses
  {$ifdef unix}clocale{$endif},
   classes, sysutils;
```

Chapter 6

Reference for unit 'cmem'

6.1 Used units

Table 6.1: Used units by unit 'cmem'

Name	Page
System	1362

6.2 Overview

The cmem memory manager sets the system units memory manager to a C-based memory manager: all memory management calls are shunted through to the C memory manager, using Malloc (535), Free (535) and ReAlloc (535). For this reason, the cmem unit should be the first unit of the uses clause of the program.

The unit also offers the C memory calls directly as external declarations from the C library, but it is recommended to use the normal FPC routines for this.

Obviously, including this unit links your program to the C library.

Remark Note that specifying the `-gv` command-line option, to enable valgrind debugging info will implicitly add this unit to your program.

6.3 Constants, types and variables

6.3.1 Constants

```
LibName = 'c'
```

`LibName` is the name of the library that is actually used. On most systems, this is simply "libc.so".

6.4 Procedures and functions

6.4.1 CAlloc

Synopsis: Allocate memory based on item size and count.

Declaration: `function CAlloc(unitSize: PtrUInt; UnitCount: PtrUInt) : pointer`

Visibility: default

Description: `Calloc` allocates memory to hold `UnitCount` units of size `UnitSize` each. The memory is one block of memory. It returns a pointer to the newly allocated memory block.

See also: Malloc (535), Free (535), Realloc (535)

6.4.2 Free

Synopsis: Free a previously allocated block.

Declaration: `procedure Free(P: pointer)`

Visibility: default

Description: `Free` returns the memory block pointed to by `P` to the system. After `Free` was called, the pointer `P` is no longer valid.

See also: Malloc (535), ReAlloc (535)

6.4.3 Malloc

Synopsis: Malloc external declaration.

Declaration: `function Malloc(Size: PtrUInt) : Pointer`

Visibility: default

Description: `Malloc` is the external declaration of the C libraries `malloc` call. It accepts a size parameter, and returns a pointer to a memory block of the requested size or `Nil` if no more memory could be allocated.

See also: Free (535), ReAlloc (535)

6.4.4 ReAlloc

Synopsis: Reallocates a memory block.

Declaration: `function ReAlloc(P: Pointer; Size: PtrUInt) : pointer`

Visibility: default

Description: `ReAlloc` re-allocates a block of memory pointed to by `p`. The new block will have size `Size`, and as much data as was available or as much data as fits is copied from the old to the new location.

See also: Malloc (535), Free (535)

Chapter 7

Reference for unit 'collation_de'

7.1 Used units

Table 7.1: Used units by unit 'collation_de'

Name	Page
System	1362

7.2 Overview

The collation_de unit registers the German Unicode collation (de). This collation bases itself on the DUCET collation, so that collation will be included as well.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 8

Reference for unit 'collation_es'

8.1 Used units

Table 8.1: Used units by unit 'collation_es'

Name	Page
System	1362

8.2 Overview

The collation_es unit registers the Spanish Unicode collation (de). This collation bases itself on the DUCET collation, so that collation will be included as well.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 9

Reference for unit 'collation_fr_ca'

9.1 Used units

Table 9.1: Used units by unit 'collation_fr_ca'

Name	Page
System	1362

9.2 Overview

The collation_fr_ca unit registers the French Unicode collation (fr). This collation bases itself on the DUCET collation, so that collation will be included as well.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 10

Reference for unit 'collation_ja'

10.1 Used units

Table 10.1: Used units by unit 'collation_ja'

Name	Page
System	1362

10.2 Overview

The collation_ja unit registers the Japanese Unicode collation (ja). This collation bases itself on the DUCET collation, so that collation will be included as well.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 11

Reference for unit 'collation_ko'

11.1 Used units

Table 11.1: Used units by unit 'collation_ko'

Name	Page
System	1362

11.2 Overview

The collation_ko unit registers the Korean Unicode collation (ko). This collation bases itself on the DUCET collation, so that collation will be included as well.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 12

Reference for unit 'collation_ru'

12.1 Used units

Table 12.1: Used units by unit 'collation_ru'

Name	Page
System	1362

12.2 Overview

The collation_ru unit registers the Russian Unicode collation (ru). This collation bases itself on the DUCET collation, so that collation will be included as well.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 13

Reference for unit 'collation_sv'

13.1 Used units

Table 13.1: Used units by unit 'collation_sv'

Name	Page
System	1362

13.2 Overview

The collation_sv unit registers the Swedish Unicode collation (sv). This collation bases itself on the DUCET collation, so that collation will be included as well.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 14

Reference for unit 'collation_zh'

14.1 Used units

Table 14.1: Used units by unit 'collation_zh'

Name	Page
System	1362

14.2 Overview

The collation_zh unit registers the Chinese Unicode collation (zh). This collation bases itself on the DUCET collation, so that collation will be included as well.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 15

Reference for unit 'cp1250'

15.1 Used units

Table 15.1: Used units by unit 'cp1250'

Name	Page
System	1362

15.2 Overview

The `cp1250` unit registers single-byte codepage 1250. This is necessary to convert single-byte strings using codepage 1250 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 16

Reference for unit 'cp1251'

16.1 Used units

Table 16.1: Used units by unit 'cp1251'

Name	Page
System	1362

16.2 Overview

The `cp1251` unit registers single-byte codepage 1251. This is necessary to convert single-byte strings using codepage 1251 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 17

Reference for unit 'cp1252'

17.1 Used units

Table 17.1: Used units by unit 'cp1252'

Name	Page
System	1362

17.2 Overview

The `cp1252` unit registers single-byte codepage 1252. This is necessary to convert single-byte strings using codepage 1252 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 18

Reference for unit 'cp1253'

18.1 Used units

Table 18.1: Used units by unit 'cp1253'

Name	Page
System	1362

18.2 Overview

The `cp1253` unit registers single-byte codepage 1253. This is necessary to convert single-byte strings using codepage 1253 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 19

Reference for unit 'cp1254'

19.1 Used units

Table 19.1: Used units by unit 'cp1254'

Name	Page
System	1362

19.2 Overview

The `cp1254` unit registers single-byte codepage 1254. This is necessary to convert single-byte strings using codepage 1254 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 20

Reference for unit 'cp1255'

20.1 Used units

Table 20.1: Used units by unit 'cp1255'

Name	Page
System	1362

20.2 Overview

The `cp1255` unit registers single-byte codepage 1255. This is necessary to convert single-byte strings using codepage 1255 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 21

Reference for unit 'cp1256'

21.1 Used units

Table 21.1: Used units by unit 'cp1256'

Name	Page
System	1362

21.2 Overview

The `cp1256` unit registers single-byte codepage 1256. This is necessary to convert single-byte strings using codepage 1256 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 22

Reference for unit 'cp1257'

22.1 Used units

Table 22.1: Used units by unit 'cp1257'

Name	Page
System	1362

22.2 Overview

The `cp1257` unit registers single-byte codepage 1257. This is necessary to convert single-byte strings using codepage 1257 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 23

Reference for unit 'cp1258'

23.1 Used units

Table 23.1: Used units by unit 'cp1258'

Name	Page
System	1362

23.2 Overview

The `cp1258` unit registers single-byte codepage 1258. This is necessary to convert single-byte strings using codepage 1258 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 24

Reference for unit 'cp437'

24.1 Used units

Table 24.1: Used units by unit 'cp437'

Name	Page
System	1362

24.2 Overview

The `cp437` unit registers single-byte codepage 437. This is necessary to convert single-byte strings using codepage 437 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 25

Reference for unit 'cp646'

25.1 Used units

Table 25.1: Used units by unit 'cp646'

Name	Page
System	1362

25.2 Overview

The `cp646` unit registers single-byte codepage 646. This is necessary to convert single-byte strings using codepage 646 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 26

Reference for unit 'cp850'

26.1 Used units

Table 26.1: Used units by unit 'cp850'

Name	Page
System	1362

26.2 Overview

The `cp850` unit registers single-byte codepage 850. This is necessary to convert single-byte strings using codepage 850 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 27

Reference for unit 'cp852'

27.1 Used units

Table 27.1: Used units by unit 'cp852'

Name	Page
System	1362

27.2 Overview

The `cp852` unit registers single-byte codepage 852. This is necessary to convert single-byte strings using codepage 852 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 28

Reference for unit 'cp856'

28.1 Used units

Table 28.1: Used units by unit 'cp856'

Name	Page
System	1362

28.2 Overview

The `cp856` unit registers single-byte codepage 856. This is necessary to convert single-byte strings using codepage 856 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 29

Reference for unit 'cp866'

29.1 Used units

Table 29.1: Used units by unit 'cp866'

Name	Page
System	1362

29.2 Overview

The `cp866` unit registers single-byte codepage 866. This is necessary to convert single-byte strings using codepage 866 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 30

Reference for unit 'cp874'

30.1 Used units

Table 30.1: Used units by unit 'cp874'

Name	Page
System	1362

30.2 Overview

The `cp874` unit registers single-byte codepage 874. This is necessary to convert single-byte strings using codepage 874 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 31

Reference for unit 'cp8859_1'

31.1 Used units

Table 31.1: Used units by unit 'cp8859_1'

Name	Page
System	1362

31.2 Overview

The `cp8859_1` unit registers single-byte codepage 8859-1. This is necessary to convert single-byte strings using codepage 8859-1 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 32

Reference for unit 'cp8859_2'

32.1 Used units

Table 32.1: Used units by unit 'cp8859_2'

Name	Page
System	1362

32.2 Overview

The `cp8859_2` unit registers single-byte codepage 8859-2. This is necessary to convert single-byte strings using codepage 8859-2 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 33

Reference for unit 'cp8859_5'

33.1 Used units

Table 33.1: Used units by unit 'cp8859_5'

Name	Page
System	1362

33.2 Overview

The `cp8859_5` unit registers single-byte codepage 8859-5. This is necessary to convert single-byte strings using codepage 8859-5 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 34

Reference for unit 'cp895'

34.1 Used units

Table 34.1: Used units by unit 'cp895'

Name	Page
System	1362

34.2 Overview

The `cp895` unit registers single-byte codepage 895. This is necessary to convert single-byte strings using codepage 895 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 35

Reference for unit 'cp932'

35.1 Used units

Table 35.1: Used units by unit 'cp932'

Name	Page
System	1362

35.2 Overview

The `cp932` unit registers single-byte codepage 932. This is necessary to convert single-byte strings using codepage 932 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 36

Reference for unit 'cp936'

36.1 Used units

Table 36.1: Used units by unit 'cp936'

Name	Page
System	1362

36.2 Overview

The `cp936` unit registers single-byte codepage 936. This is necessary to convert single-byte strings using codepage 936 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 37

Reference for unit 'cp949'

37.1 Used units

Table 37.1: Used units by unit 'cp949'

Name	Page
System	1362

37.2 Overview

The `cp949` unit registers single-byte codepage 949. This is necessary to convert single-byte strings using codepage 949 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 38

Reference for unit 'cp950'

38.1 Used units

Table 38.1: Used units by unit 'cp950'

Name	Page
System	1362

38.2 Overview

The `cp950` unit registers single-byte codepage 950. This is necessary to convert single-byte strings using codepage 950 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 39

Reference for unit 'cpall'

39.1 Used units

Table 39.1: Used units by unit 'cpall'

Name	Page
cp1250	544
cp1251	545
cp1252	546
cp1253	547
cp1254	548
cp1255	549
cp1256	550
cp1257	551
cp1258	552
cp3021	??
cp437	553
cp646	554
cp737	??
cp775	??
cp850	555
cp852	556
cp855	??
cp856	557
cp857	??
cp860	??
cp861	??
cp862	??
cp863	??
cp864	??
cp865	??
cp866	558
cp869	??
cp874	559
cp8859_1	560
cp8859_10	??
cp8859_11	??
cp8859_13	??
cp8859_14	??
cp8859_15	??
cp8859_16	??
cp8859_2	561
cp8859_3	??
cp8859_4	??
cp8859_5	562

39.2 Overview

The `cpall` unit registers all known single-byte codepages: 1251 (569), 866 (569), ISO 8856-5 (569) (cyrillic), 8859-1 (569), 8859-2 (569), 1253 (569) (greek), 850 (569), 437 (569), 1252 (569), 646 (569), 874 (569), 856 (569), 1250 (569), 1254 (569), 1255 (569), 1256 (569), 1257 (569), 1258 (569), 852 (569).

This unit does not contain any routines. It simply uses the other units so all corresponding code pages are registered.

Chapter 40

Reference for unit 'Crt'

40.1 Used units

Table 40.1: Used units by unit 'Crt'

Name	Page
System	1362

40.2 Overview

This chapter describes the `CRT` unit for Free Pascal, under all of Dos, Linux and Windows. The unit was first written for Dos by Florian Klaempfl. The unit was ported to Linux by Mark May and enhanced by Michael Van Canneyt and Peter Vreman. It works on the Linux console, and in xterm and rxvt windows under X-Windows. The functionality for both is the same, except that under Linux the use of an early implementation (versions 0.9.1 and earlier of the compiler) the CRT unit automatically cleared the screen at program startup.

There are some caveats when using the CRT unit:

- Programs using the CRT unit will *not* be usable when input/output is being redirected on the command-line.
- For similar reasons they are not usable as CGI-scripts for use with a webserver.
- The use of the CRT unit and the graph unit may not always be supported.
- The CRT unit is not thread safe.
- On Linux or other Unix OSes, executing other programs that expect special terminal behaviour (using one of the special functions in the Linux unit) will not work. The terminal is set in RAW mode, which will destroy most terminal emulation settings.
- The CRT unit stems from the TP/Dos area. It is designed to work with single-byte character sets, where 1 char = 1 byte. That means that widestrings or UTF-8 encoded (ansi)strings will not work correctly.

40.3 Constants, types and variables

40.3.1 Constants

```
Black = 0
```

Black color attribute.

```
Blink = 128
```

Blink attribute.

```
Blue = 1
```

Blue color attribute.

```
Brown = 6
```

Brown color attribute.

```
BW40 = 0
```

40 columns black and white screen mode.

```
BW80 = 2
```

80 columns black and white screen mode.

```
C40 = CO40
```

40 columns color screen mode.

```
C80 = CO80
```

80 columns color screen mode.

```
CO40 = 1
```

40 columns color screen mode.

```
CO80 = 3
```

80 columns color screen mode.

```
ConsoleMaxX = 1024
```

```
ConsoleMaxY = 1024
```

```
Cyan = 3
```

Cyan color attribute.

```
DarkGray = 8
```

Dark gray color attribute.

```
Flushing = False
```

```
Font8x8 = 256
```

Internal ROM font mode.

```
Green = 2
```

Green color attribute.

```
LightBlue = 9
```

Light Blue color attribute.

```
LightCyan = 11
```

Light cyan color attribute.

```
LightGray = 7
```

Light gray color attribute.

```
LightGreen = 10
```

Light green color attribute.

```
LightMagenta = 13
```

Light magenta color attribute.

```
LightRed = 12
```

Light red color attribute.

```
Magenta = 5
```

Magenta color attribute.

```
Mono = 7
```

Monochrome screen mode (hercules screens).

```
Red = 4
```

Red color attribute.

```
ScreenHeight : LongInt = 25
```

Current screen height.

```
ScreenWidth : LongInt = 80
```

Current screen width.

```
White = 15
```

White color attribute.

```
Yellow = 14
```

Yellow color attribute.

40.3.2 Types

```
tcrtcoord = 1..255
```

`tcrtcoord` is a subrange type for denoting CRT coordinates. It supports coordinates ranging from 1 to 255. Using this type together with range-checking turned on can be used to debug CRT code.

40.3.3 Variables

```
CheckBreak : Boolean
```

Check for CTRL-Break keystroke. Not used.

```
CheckEOF : Boolean
```

Check for EOF on standard input. Not used.

```
CheckSnow : Boolean
```

Check snow on CGA screens. Not used.

```
ConsoleBuf : PConsoleBuf
```

```
DirectVideo : Boolean
```

The `DirectVideo` variable controls the writing to the screen. If it is `True`, the cursor is set via direct port access. If `False`, then the BIOS is used. This is defined under dos only.

```
LastMode : Word = 3
```

The `Lastmode` variable tells you which mode was last selected for the screen. It is defined on DOS only.

```
TextAttr : Byte = $07
```

The `TextAttr` variable controls the attributes with which characters are written to screen.

```
WindMax : Word = $184f
```

The upper byte of `WindMax` contains the Y coordinate while the lower byte contains the X coordinate. The use of this variable is deprecated, use `WindMaxX` and `WindMaxY` instead.

```
WindMaxX : DWord
```

X coordinate of lower right corner of the defined window.

```
WindMaxY : DWord
```

Y coordinate of lower right corner of the defined window.

```
WindMin : Word = $0
```

The upper byte of `WindMin` contains the Y coordinate while the lower byte contains the X coordinate. The use of this variable is deprecated, use `WindMinX` and `WindMinY` instead.

```
WindMinX : DWord
```

X coordinate of upper left corner of the defined window.

```
WindMinY : DWord
```

Y coordinate of upper left corner of the defined window.

40.4 Procedures and functions

40.4.1 AssignCrt

Synopsis: Assign file to CRT.

Declaration: `procedure AssignCrt(var F: Text)`

Visibility: default

Description: `AssignCrt` Assigns a file `F` to the console. Everything written to the file `F` goes to the console instead. If the console contains a window, everything is written to the window instead.

Errors: None.

See also: Window (586)

Listing: ./examples/crtex/ex1.pp

```
Program Example1;
uses Crt;

{ Program to demonstrate the AssignCrt function. }

var
  F : Text;
begin
  AssignCrt(F);
  Rewrite(F); { Don't forget to open for output! }
  WriteLn(F,'This is written to the Assigned File');
  Close(F);
end.
```

40.4.2 ClrEol

Synopsis: Clear from cursor position till end of line.

Declaration: `procedure ClrEol`

Visibility: default

Description: `ClrEol` clears the current line, starting from the cursor position, to the end of the window. The cursor doesn't move

Errors: None.

See also: DelLine (578), InsLine (580), ClrScr (577)

Listing: ./examples/crtex/ex9.pp

```
Program Example9;
uses Crt;

{ Program to demonstrate the ClrEol function. }
var
  I,J : integer;

begin
  For I:=1 to 15 do
    For J:=1 to 80 do
      begin
      gotoxy(j,i);
      Write(j mod 10);
      end;
  Window(5,5,75,12);
  Write('This line will be cleared from',
        ' here till the right of the window');
  GotoXY(27,WhereY);
  ReadKey;
  ClrEol;
  WriteLn;
end.
```

40.4.3 ClrScr

Synopsis: Clear current window.

Declaration: `procedure ClrScr`

Visibility: default

Description: `ClrScr` clears the current window (using the current colors), and sets the cursor in the top left corner of the current window.

Errors: None.

See also: Window (586)

Listing: ./examples/crtex/ex8.pp

```
Program Example8;
uses Crt;

{ Program to demonstrate the ClrScr function. }

begin
  WriteLn('Press any key to clear the screen');
  ReadKey;
  ClrScr;
  WriteLn('Have fun with the cleared screen');
end.
```

40.4.4 cursorbig

Synopsis: Show big cursor.

Declaration: `procedure cursorbig`

Visibility: default

Description: `CursorBig` makes the cursor a big rectangle. Not implemented on unixes.

Errors: None.

See also: CursorOn (578), CursorOff (577)

40.4.5 cursoroff

Synopsis: Hide cursor.

Declaration: `procedure cursoroff`

Visibility: default

Description: `CursorOff` switches the cursor off (i.e. the cursor is no longer visible). Not implemented on unixes.

Errors: None.

See also: CursorOn (578), CursorBig (577)

40.4.6 cursoron

Synopsis: Display cursor.

Declaration: `procedure cursoron`

Visibility: default

Description: `CursorOn` switches the cursor on. Not implemented on unixes.

Errors: None.

See also: CursorBig (577), CursorOff (577)

40.4.7 Delay

Synopsis: Delay program execution.

Declaration: `procedure Delay(MS: Word)`

Visibility: default

Description: `Delay` waits a specified number of milliseconds. The number of specified seconds is an approximation, and may be off a lot, if system load is high.

Errors: None

See also: Sound (583), NoSound (582)

Listing: ./examples/crtex/ex15.pp

```
Program Example15;
uses Crt;

{ Program to demonstrate the Delay function. }
var
  i : longint;
begin
  WriteLn('Counting Down');
  for i:=10 downto 1 do
   begin
     WriteLn(i);
     Delay(1000); {Wait one second}
   end;
  WriteLn('BOOM!!!');
end.
```

40.4.8 DelLine

Synopsis: Delete line at cursor position.

Declaration: `procedure DelLine`

Visibility: default

Description: `DelLine` removes the current line. Lines following the current line are scrolled 1 line up, and an empty line is inserted at the bottom of the current window. The cursor doesn't move.

Errors: None.

See also: ClrEol (576), InsLine (580), ClrScr (577)

Listing: ./examples/crtex/ex11.pp

```
Program Example10;
uses Crt;

{ Program to demonstrate the InsLine function. }

begin
  ClrScr;
  WriteLn;
  WriteLn('Line 1');
  WriteLn('Line 2');
  WriteLn('Line 2');
  WriteLn('Line 3');
  WriteLn;
  WriteLn('Oops, Line 2 is listed twice,',
          ' let''s delete the line at the cursor postion');
  GotoXY(1,3);
  ReadKey;
  DelLine;
  GotoXY(1,10);
end.
```

40.4.9 GotoXY

Synopsis: Set cursor position on screen.

Declaration: `procedure GotoXY(X: tcrtcoord; Y: tcrtcoord)`

Visibility: default

Description: `GotoXY` positions the cursor at `(X,Y)`, `X` in horizontal, `Y` in vertical direction relative to the origin of the current window. The origin is located at `(1,1)`, the upper-left corner of the window.

Errors: None.

See also: WhereX (585), WhereY (585), Window (586)

Listing: ./examples/crtex/ex6.pp

```
Program Example6;
uses Crt;

{ Program to demonstrate the GotoXY function. }

begin
  ClrScr;
  GotoXY(10,10);
  Write('10,10');
  GotoXY(70,20);
  Write('70,20');
  GotoXY(1,22);
end.
```

40.4.10 HighVideo

Synopsis: Switch to highlighted text mode.

Declaration: `procedure HighVideo`

Visibility: default

Description: `HighVideo` switches the output to highlighted text. (It sets the high intensity bit of the video attribute)

Errors: None.

See also: TextColor (584), TextBackground (583), LowVideo (581), NormVideo (582)

Listing: ./examples/crtex/ex14.pp

```
Program Example14;
uses Crt;

{ Program to demonstrate the LowVideo, HighVideo, NormVideo functions. }

begin
  LowVideo;
  WriteLn('This is written with LowVideo');
  HighVideo;
  WriteLn('This is written with HighVideo');
  NormVideo;
  WriteLn('This is written with NormVideo');
end.
```

40.4.11 InsLine

Synopsis: Insert an empty line at cursor position.

Declaration: `procedure InsLine`

Visibility: default

Description: `InsLine` inserts an empty line at the current cursor position. Lines following the current line are scrolled 1 line down, causing the last line to disappear from the window. The cursor doesn't move.

Errors: None.

See also: ClrEol (576), DelLine (578), ClrScr (577)

Listing: ./examples/crtex/ex10.pp

```
Program Example10;
uses Crt;

{ Program to demonstrate the InsLine function. }

begin
  ClrScr;
  WriteLn;
  WriteLn('Line 1');
  WriteLn('Line 3');
  WriteLn;
```

```
  WriteLn('Oops, forgot Line 2, let''s insert at the cursor postion');
  GotoXY(1,3);
  ReadKey;
  InsLine;
  Write('Line 2');
  GotoXY(1,10);
end.
```

40.4.12 KeyPressed

Synopsis: Check if there is a keypress in the keybuffer.

Declaration: `function KeyPressed : Boolean`

Visibility: default

Description: `Keypressed` scans the keyboard buffer and sees if a key has been pressed. If this is the case, `True` is returned. If not, `False` is returned. The `Shift, Alt, Ctrl` keys are not reported. The key is not removed from the buffer, and can hence still be read after the `KeyPressed` function has been called.

Errors: None.

See also: ReadKey (582)

Listing: ./examples/crtex/ex2.pp

```
Program Example2;
uses Crt;

{ Program to demonstrate the KeyPressed function. }

begin
  WriteLn('Waiting until a key is pressed');
  repeat
  until KeyPressed;
 { The key is not Read,
   so it should also be outputted at the commandline}
end.
```

40.4.13 LowVideo

Synopsis: Switch to low intensity colors.

Declaration: `procedure LowVideo`

Visibility: default

Description: `LowVideo` switches the output to non-highlighted text. (It clears the high intensity bit of the video attribute)

For an example, see HighVideo (580)

Errors: None.

See also: TextColor (584), TextBackground (583), HighVideo (580), NormVideo (582)

40.4.14 NormVideo

Synopsis: Return to normal (startup) modus.

Declaration: `procedure NormVideo`

Visibility: default

Description: `NormVideo` switches the output to the defaults, read at startup. (The defaults are read from the cursor position at startup)

For an example, see HighVideo (580)

Errors: None.

See also: TextColor (584), TextBackground (583), LowVideo (581), HighVideo (580)

40.4.15 NoSound

Synopsis: Stop system speaker.

Declaration: `procedure NoSound`

Visibility: default

Description: `NoSound` stops the speaker sound. This call is not supported on all operating systems.

Errors: None.

See also: Sound (583)

Listing: ./examples/crtex/ex16.pp

```
Program Example16;
uses Crt;

{ Program to demonstrate the Sound and NoSound function. }

var
  i : longint;
begin
  WriteLn('You will hear some tones from your speaker');
  i:=0;
  while (i<15000) do
   begin
     inc(i,500);
     Sound(i);
     Delay(100);
   end;
  WriteLn('Quiet now!');
  NoSound; {Stop noise}
end.
```

40.4.16 ReadKey

Synopsis: Read key from keybuffer.

Declaration: `function ReadKey : Char`

Visibility: default

Description: `ReadKey` reads 1 key from the keyboard buffer, and returns this. If an extended or function key has been pressed, then the zero ASCII code is returned. You can then read the scan code of the key with a second ReadKey call.

Key mappings under Linux can cause the wrong key to be reported by `ReadKey`, so caution is needed when using `ReadKey`.

Errors: None.

See also: KeyPressed (581)

Listing: ./examples/crtex/ex3.pp

```
Program Example3;
uses Crt;

{ Program to demonstrate the ReadKey function. }

var
  ch : char;
begin
  writeln('Press Left/Right, Esc=Quit');
  repeat
    ch:=ReadKey;
    case ch of
     #0 : begin
            ch:=ReadKey; {Read ScanCode}
            case ch of
             #75 : WriteLn('Left');
             #77 : WriteLn('Right');
            end;
          end;
    #27 : WriteLn('ESC');
    end;
  until ch=#27 {Esc}
end.
```

40.4.17 Sound

Synopsis: Sound system speaker.

Declaration: `procedure Sound(Hz: Word)`

Visibility: default

Description: `Sound` sounds the speaker at a frequency of `hz`. Under Windows, a system sound is played and the frequency parameter is ignored. On other operating systems, this routine may not be implemented.

Errors: None.

See also: NoSound (582)

40.4.18 TextBackground

Synopsis: Set text background.

Declaration: `procedure TextBackground(Color: Byte)`

Visibility: default

Description: `TextBackground` sets the background color to `CL`. `CL` can be one of the predefined color constants.

Errors: None.

See also: TextColor (584), HighVideo (580), LowVideo (581), NormVideo (582)

Listing: ./examples/crtex/ex13.pp

```
Program Example13;
uses Crt;

{ Program to demonstrate the TextBackground function. }

begin
  TextColor(White);
  WriteLn('This is written in with the default background color');
  TextBackground(Green);
  WriteLn('This is written in with a Green background');
  TextBackground(Brown);
  WriteLn('This is written in with a Brown background');
  TextBackground(Black);
  WriteLn('Back with a black background');
end.
```

40.4.19 TextColor

Synopsis: Set text color.

Declaration: `procedure TextColor(Color: Byte)`

Visibility: default

Description: `TextColor` sets the foreground color to `CL`. `CL` can be one of the predefined color constants.

Errors: None.

See also: TextBackground (583), HighVideo (580), LowVideo (581), NormVideo (582)

Listing: ./examples/crtex/ex12.pp

```
Program Example12;
uses Crt;

{ Program to demonstrate the TextColor function. }

begin
  WriteLn('This is written in the default color');
  TextColor(Red);
  WriteLn('This is written in Red');
  TextColor(White);
  WriteLn('This is written in White');
  TextColor(LightBlue);
  WriteLn('This is written in Light Blue');
end.
```

40.4.20 TextMode

Synopsis: Set screen mode.

Declaration: `procedure TextMode(Mode: Word)`

Visibility: default

Description: `TextMode` sets the textmode of the screen (i.e. the number of lines and columns of the screen). The lower byte is use to set the VGA text mode.

This procedure is only implemented on dos.

Errors: None.

See also: Window (586)

40.4.21 WhereX

Synopsis: Return X (horizontal) cursor position.

Declaration: `function WhereX : tcrtcoord`

Visibility: default

Description: `WhereX` returns the current X-coordinate of the cursor, relative to the current window. The origin is `(1,1)`, in the upper-left corner of the window.

Errors: None.

See also: GotoXY (579), WhereY (585), Window (586)

Listing: ./examples/crtex/ex7.pp

```
Program Example7;
uses Crt;

{ Program to demonstrate the WhereX and WhereY functions. }

begin
  WriteLn('Cursor postion: X=',WhereX,' Y=',WhereY);
end.
```

40.4.22 WhereY

Synopsis: Return Y (vertical) cursor position.

Declaration: `function WhereY : tcrtcoord`

Visibility: default

Description: `WhereY` returns the current Y-coordinate of the cursor, relative to the current window. The origin is `(1,1)`, in the upper-left corner of the window.

Errors: None.

See also: GotoXY (579), WhereX (585), Window (586)

Listing: ./examples/crtex/ex7.pp

```
Program Example7;
uses Crt;

{ Program to demonstrate the WhereX and WhereY functions. }

begin
  WriteIn('Cursor postion: X=',WhereX,' Y=',WhereY);
end.
```

40.4.23 Window

Synopsis: Create new window on screen.

Declaration: `procedure Window(X1: Byte; Y1: Byte; X2: Byte; Y2: Byte)`

Visibility: default

Description: `Window` creates a window on the screen, to which output will be sent. `(X1,Y1)` are the coordinates of the upper left corner of the window, `(X2,Y2)` are the coordinates of the bottom right corner of the window. These coordinates are relative to the entire screen, with the top left corner equal to `(1,1)`. Further coordinate operations, except for the next Window call, are relative to the window's top left corner.

Errors: None.

See also: GotoXY (579), WhereX (585), WhereY (585), ClrScr (577)

Listing: ./examples/crtex/ex5.pp

```
Program Example5;
uses Crt;

{ Program to demonstrate the Window function. }

begin
  ClrScr;
  WriteLn('Creating a window from 30,10 to 50,20');
  Window(30,10,50,20);
  WriteLn('We are now writing in this small window we just created, we '+
          'can''t get outside it when writing long lines like this one');
  Write('Press any key to clear the window');
  ReadKey;
  ClrScr;
  Write('The window is cleared, press any key to restore to fullscreen');
  ReadKey;
{Full Screen is 80x25}
  Window(1,1,80,25);
  Clrscr;
  WriteIn('Back in Full Screen');
end.
```

40.5 TCharAttr

```
TCharAttr = packed record
```

```
  ch : Char;
  attr : Byte;
end
```

Chapter 41

Reference for unit 'cthreads'

41.1 Used units

Table 41.1: Used units by unit 'cthreads'

Name	Page
System	1362

41.2 Overview

The CThreads unit initializes the system unit's thread management routines with an implementation based on the POSIX thread managing routines in the C library. This assures that C libraries that are thread-aware still work if they are linked to by a FPC program.

It doesn't offer any API by itself: the initialization section of the unit just initializes the `ThreadManager` record in the System (1362) unit. This is done using the SetCThreadManager (589) call

The cthreads unit simply needs to be included in the uses clause of the program, preferably the very first unit, and the initialization section of the unit will do all the work.

Note that including this unit links your program to the C library of the system.

It makes no sense to use this unit on a non-POSIX system: Windows, OS/2 or DOS, therefor it should always between an ifdef statement:

```
program myprogram;

uses
  {$ifdef unix}cthreads{$endif},
   classes, sysutils;
```

The Lazarus IDE inserts this conditional automatically for each new started program.

41.3 Procedures and functions

41.3.1 SetCThreadManager

Synopsis: Sets the thread manager to the C thread manager.

Declaration: `procedure SetCThreadManager`

Visibility: default

Description: `SetCThreadManager` actually sets the thread manager to the C thread manager. It can be called to re-set the thread manager if the thread manager was set to some other thread manager during the life-time of the program.

Chapter 42

Reference for unit 'ctypes'

42.1 Used units

Table 42.1: Used units by unit 'ctypes'

Name	Page
System	1362
unixtype	2175

42.2 Overview

The ctypes unit contains the definitions of commonly found C types. It can be used when interfaces to C libraries need to be defined. The types here are correct on all platforms, 32 or 64 bit.

The main advantage of using this file is to make sure that all C header import units use the same definitions for basic C types.

The h2pas program can include the ctypes unit automatically in the units it generates. The `-C` command-line switch can be used for this.

42.3 Constants, types and variables

42.3.1 Types

```
cbool = UnixType.cbool
```

C boolean (longbool).

```
cchar = UnixType.cchar
```

C character type (No signedness specification, 8 bit integer).

```
cdouble = UnixType.cdouble
```

Double precision floating point type (double).

```
cfloat = UnixType.cfloat
```

Single precision floating point type (single).

```
cint = UnixType.cint
```

C integer (commonly 32 bit).

```
cint16 = UnixType.cint16
```

16-bit signed integer.

```
cint32 = UnixType.cint32
```

32-bit signed integer (commonly: int).

```
cint64 = UnixType.cint64
```

64-bit integer.

```
cint8 = UnixType.cint8
```

8-bit signed integer.

```
clong = UnixType.clong
```

long integer (32/64 bit, depending on CPU register size).

```
clongdouble = Double
```

Long precision floating point type (extended/double, depending on CPU).

```
clonglong = UnixType.clonglong
```

Long (64-bit) integer.

```
coff_t = UnixType.TOff
```

Generic type to indicate offset.

```
cschar = UnixType.cschar
```

C signed character type (8 bit signed integer).

```
cshort = UnixType.cshort
```

Short integer (16 bit).

```
csigned = UnixType.csigned
```

Signed integer (commonly 32 bit).

```
csint = UnixType.csint
```

Signed integer (commonly 32 bit).

```
csize_t = UnixType.size_t
```

Generic type to contain a size of all kinds of structures.

```
cslong = UnixType.cslong
```

Signed long integer (32/64 bit, depending on CPU register size).

```
cslonglong = UnixType.cslonglong
```

Signed long (64-bit) integer.

```
csshort = UnixType.csshort
```

Short signed integer (16 bit).

```
cuchar = UnixType.cuchar
```

C unsigned character type (8 bit unsigned integer).

```
cuint = UnixType.cuint
```

Unsigned integer (commonly 32 bit).

```
cuint16 = UnixType.cuint16
```

16-bit unsigned integer.

```
cuint32 = UnixType.cuint32
```

32-bit unsigned integer.

```
cuint64 = UnixType.cuint64
```

Unsigned 64-bit integer.

```
cuint8 = UnixType.cuint8
```

8-bit unsigned integer.

```
culong = UnixType.culong
```

Unsigned long integer (32/64 bit, depending on CPU register size).

```
culonglong = UnixType.culonglong
```

Unsigned long (64-bit) integer.

```
cunsigned = UnixType.cunsigned
```

Unsigned integer (commonly 32 bit).

```
cushort = UnixType.cushort
```

Short unsigned integer (16 bit).

```
pcbool = UnixType.pcbool
```

Pointer to cbool (590) type.

```
pcchar = UnixType.pcchar
```

Pointer to cchar (590) type.

```
pcdouble = UnixType.pcdouble
```

Pointer to cdouble (590) type.

```
pcfloat = UnixType.pcfloat
```

Pointer to cfloat (591) type.

```
pcint = UnixType.pcint
```

Pointer to cint (591) type.

```
pcint16 = UnixType.pcint16
```

Pointer to cint16 (591) type.

```
pcint32 = UnixType.pcint32
```

Pointer to cint32 (591) type.

```
pcint64 = UnixType.pcint64
```

Pointer to cint64 (591) type.

```
pcint8 = UnixType.pcint8
```

Pointer to cint8 (591) type.

```
pclong = UnixType.pclong
```

Pointer to clong (591) type.

```
Pclongdouble = ^clongdouble
```

Pointer to clongdouble (591) type.

```
pclonglong = UnixType.pclonglong
```

Pointer to clonglong (591) type.

```
pcschar = UnixType.pcschar
```

Pointer to cschar (591) type.

```
pcshort = UnixType.pcshort
```

Pointer to cshort (591) type.

```
pcsigned = UnixType.pcsigned
```

Pointer to csigned (591) type.

```
pcsint = UnixType.pcsint
```

Pointer to csint (592) type.

```
pcsize_t = UnixType.psize_t
```

Pointer to generic size type.

```
pcslong = UnixType.pcslong
```

Pointer to cslong (592) type.

```
pcslonglong = UnixType.pcslonglong
```

Pointer to cslonglong (592) type.

```
pcsshort = UnixType.pcsshort
```

Pointer to csshort (592) type.

```
pcuchar = UnixType.pcuchar
```

Pointer to cuchar (592) type.

```
pcuint = UnixType.pcuint
```

Pointer to cuint (592) type.

```
pcuint16 = UnixType.pcuint16
```

Pointer to cuint16 (592) type.

```
pcuint32 = UnixType.pcuint32
```

Pointer to cuint32 (592) type.

```
pcuint64 = UnixType.pcuint64
```

Pointer to cuint64 (592) type.

```
pcuint8 = UnixType.pcuint8
```

Pointer to cuint8 (592) type.

```
pculong = UnixType.pculong
```

Pointer to culong (592) type.

```
pculonglong = UnixType.pculonglong
```

Pointer to culonglong (592) type.

```
pcunsigned = UnixType.pcunsigned
```

Pointer to cunsigned (593) type.

```
pcushort = UnixType.pcushort
```

Pointer to cushort (593) type.

Chapter 43

Reference for unit 'cwstring'

43.1 Used units

Table 43.1: Used units by unit 'cwstring'

Name	Page
System	1362

43.2 Overview

The `cwstring` unit offers no API by itself: it just initializes the widestring manager record of the system (1362) unit with an implementation that uses collation and conversion routines which are provided by the C library found on most Unix or Linux systems that are POSIX compliant.

The cwstring should simply be included in the uses clause of the program, preferably as one of the first units, and the initialization section of the unit will do all the work.

Note that including this unit links your program to the C library of the system.

It makes no sense to use this unit on a non-POSIX system like Windows, OS/2 or DOS. Therefor it should always be enclosed with an ifdef statement:

```
program myprogram;

uses
  {$ifdef unix}cwstring,{$endif}
   classes, sysutils;
```

43.3 Procedures and functions

43.3.1 SetCWidestringManager

Synopsis: Set the Widestring manager of the system unit to the C version.

Declaration: `procedure SetCWidestringManager`

Visibility: default

Description: `SetCWidestringManager` actually sets the widestring manager record of the system unit. It is called automatically by the initialization section of the unit.

Chapter 44

Reference for unit 'DateUtils'

44.1 Used units

Table 44.1: Used units by unit 'DateUtils'

Name	Page
Math	1008
System	1362
sysutils	1635

44.2 Overview

DateUtils contains a large number of date/time manipulation routines, all based on the `TDateTime` type. There are routines for date/time math, for comparing dates and times, for composing dates and decomposing dates in their constituent parts.

44.3 Constants, types and variables

44.3.1 Constants

```
ApproxDaysPerMonth : Double = 30.4375
```

Average number of days in a month, measured over a year. Used in MonthsBetween (649).

```
ApproxDaysPerYear : Double = 365.25
```

Average number of days in a year, measured over 4 years. Used in YearsBetween (693).

```
DayFriday = 5
```

ISO day number for Friday.

```
DayMonday = 1
```

ISO day number for Monday.

```
DaySaturday = 6
```

ISO day number for Saturday.

```
DaysPerWeek = 7
```

Number of days in a week.

```
DaysPerYear : Array[Boolean] of Word = (365, 366)
```

Array with number of days in a year. The Boolean index indicates whether it is a leap year or not.

```
DaySunday = 7
```

ISO day number for Sunday.

```
DayThursday = 4
```

ISO day number for Thursday.

```
DayTuesday = 2
```

ISO day number for Tuesday.

```
DayWednesday = 3
```

ISO day number for Wednesday.

```
MonthApril = 4
```

ISO month number for April.

```
MonthAugust = 8
```

ISO month number for August.

```
MonthDecember = 12
```

ISO month number for December.

```
MonthFebruary = 2
```

ISO month number for february.

```
MonthJanuary = 1
```

ISO month number for january.

```
MonthJuly = 7
```

ISO month number for July.

```
MonthJune = 6
```

ISO month number for June.

```
MonthMarch = 3
```

ISO month number for March.

```
MonthMay = 5
```

ISO month number for May.

```
MonthNovember = 11
```

ISO month number for November.

```
MonthOctober = 10
```

ISO month number for October.

```
MonthSeptember = 9
```

ISO month number for September.

```
MonthsPerYear = 12
```

Number of months in a year.

```
OneHour = TDateTime(1) / HoursPerDay
```

One hour as a fraction of a day (suitable for TDateTime).

```
OneMillisecond = TDateTime(1) / MSecsPerDay
```

One millisecond as a fraction of a day (suitable for TDateTime).

```
OneMinute = TDateTime(1) / MinsPerDay
```

One minute as a fraction of a day (suitable for TDateTime).

```
OneSecond = TDateTime(1) / SecsPerDay
```

One second as a fraction of a day (suitable for TDateTime).

```
RecodeLeaveFieldAsIs = High(Word)
```

Bitmask deciding what to do with each TDateTime field in recode routines.

```
WeeksPerFortnight = 2
```

Number of weeks in fortnight.

`YearsPerCentury = 100`

Number of years in a century.

`YearsPerDecade = 10`

Number of years in a decade.

`YearsPerMillennium = 1000`

Number of years in a millennium.

44.4 Procedures and functions

44.4.1 CompareDate

Synopsis: Compare 2 dates, disregarding the time of day.

Declaration:
```
function CompareDate(const A: TDateTime; const B: TDateTime)
                          : TValueRelationship
```

Visibility: default

Description: `CompareDate` compares the date parts of two timestamps `A` and `B` and returns the following results:

< 0 if the day part of `A` is earlier than the day part of `B`.

0 if `A` and `B` are the on same day (times may differ) .

> 0 if the day part of `A` is later than the day part of `B`.

See also: CompareTime (603), CompareDateTime (602), SameDate (658), SameTime (660), SameDateTime (659)

Listing: ./examples/datutex/ex99.pp

```
Program Example99;

{ This program demonstrates the CompareDate function }

Uses SysUtils, DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy';

Procedure Test(D1,D2 : TDateTime);

Var
  Cmp : Integer;

begin
  Write(FormatDateTime(Fmt,D1),' is ');
  Cmp:=CompareDate(D1,D2);
  If Cmp<0 then
```

```
    write('earlier than ')
  else if Cmp>0 then
    Write('later than ')
  else
    Write('equal to ');
  WriteIn(FormatDateTime(Fmt,D2));
end;

Var
  D,N : TDateTime;

Begin
  D:=Today;
  N:=Now;
  Test(D,D);
  Test(N,N);
  Test(D+1,D);
  Test(D-1,D);
  Test(D+OneSecond,D);
  Test(D-OneSecond,D);
  Test(N+OneSecond,N);
  Test(N-OneSecond,N);
End.
```

44.4.2 CompareDateTime

Synopsis: Compare 2 dates, taking into account the time of day.

Declaration:
```
function CompareDateTime(const A: TDateTime; const B: TDateTime)
                         : TValueRelationship
```

Visibility: default

Description: `CompareDateTime` compares two timestamps `A` and `B` and returns the following results:

< 0 if `A` is earlier in date/time than `B`.

0 if `A` and `B` are the same date/time .

> 0 if `A` is later in date/time than `B`.

See also: CompareTime (603), CompareDate (601), SameDate (658), SameTime (660), SameDateTime (659)

Listing: ./examples/datutex/ex98.pp

```
Program Example98;

{ This program demonstrates the CompareDateTime function }

Uses SysUtils,DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss.zzz';

Procedure Test(D1,D2 : TDateTime);

Var
  Cmp : Integer;
```

```
begin
  Write(FormatDateTime(Fmt,D1),' is ');
  Cmp:=CompareDateTime(D1,D2);
  If Cmp<0 then
    write('earlier than ')
  else if Cmp>0 then
    Write('later than ')
  else
    Write('equal to ');
  WriteIn(FormatDateTime(Fmt,D2));
end;

Var
  D,N : TDateTime;

Begin
  D:=Today;
  N:=Now;
  Test(D,D);
  Test(N,N);
  Test(D+1,D);
  Test(D-1,D);
  Test(D+OneSecond,D);
  Test(D-OneSecond,D);
  Test(N+OneSecond,N);
  Test(N-OneSecond,N);
End.
```

44.4.3 CompareTime

Synopsis: Compares two times of the day, disregarding the date part.

Declaration:
```
function CompareTime(const A: TDateTime; const B: TDateTime)
                          : TValueRelationship
```

Visibility: default

Description: `CompareTime` compares the time parts of two timestamps `A` and `B` and returns the following results:

< 0 if the time part of `A` is earlier than the time part of `B`.

0 if `A` and `B` have the same time part (dates may differ) .

> 0 if the time part of `A` is later than the time part of `B`.

See also: CompareDateTime (602), CompareDate (601), SameDate (658), SameTime (660), SameDateTime (659)

Listing: ./examples/datutex/ex100.pp

```
Program Example100;

{ This program demonstrates the CompareTime function }

Uses SysUtils, DateUtils;
```

```
Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss.zzz';

Procedure Test(D1,D2 : TDateTime);

Var
  Cmp : Integer;

begin
  Write(FormatDateTime(Fmt,D1),' has ');
  Cmp:=CompareDateTime(D1,D2);
  If Cmp<0 then
    write('earlier time than ')
  else if Cmp>0 then
    Write('later time than ')
  else
    Write('equal time with ');
  WriteIn(FormatDateTime(Fmt,D2));
end;

Var
  D,N : TDateTime;

Begin
  D:=Today;
  N:=Now;
  Test(D,D);
  Test(N,N);
  Test(N+1,N);
  Test(N-1,N);
  Test(N+OneSecond,N);
  Test(N-OneSecond,N);
End.
```

44.4.4 DateInRange

Synopsis: Checks whether a date value is in a given rang.

Declaration:
```
function DateInRange(ADate: TDate; AStartDate: TDate; AEndDate: TDate;
                     AInclusive: Boolean) : Boolean
```

Visibility: default

Description: `DateInRange` checks whether the value `ADate` lies between `AStartDate` and `AEndDate`, and returns `True` if it is. When `AInclusive` is `True` (the default), then the limits are included. When `AInclusive` is `False`, the limits are excluded. Only the date part of the 3 parameters is considered.

Errors: The `AStartDate` value must be before `AEndDate`, but no check is performed.

See also: TimeInRange (670), DateTimeInRange (605)

44.4.5 DateOf

Synopsis: Extract the date part from a TDateTime indication.

Declaration:
```
function DateOf(const AValue: TDateTime) : TDateTime
```

Visibility: default

Description: `DateOf` extracts the date part from `AValue` and returns the result.

Since the `TDateTime` is actually a double with the date part encoded in the integer part, this operation corresponds to a call to Int (598).

See also: TimeOf (670), YearOf (693), MonthOf (648), DayOf (607), HourOf (622), MinuteOf (644), SecondOf (661), MilliSecondOf (640)

Listing: ./examples/datutex/ex1.pp

```
Program Example1;

{ This program demonstrates the DateOf function }

Uses SysUtils, DateUtils;

Begin
  WriteIn('Date is: ',DateTimeToStr(DateOf(Now)));
End.
```

44.4.6 DateTimeInRange

Synopsis: Checks whether a date/time value is in a given range.

Declaration:
```
function DateTimeInRange(ADateTime: TDateTime;
                         AStartDateTime: TDateTime;
                         AEndDateTime: TDateTime; aInclusive: Boolean)
                          : Boolean
```

Visibility: default

Description: `DateTimeInRange` checks whether the value `ADateTime` lies between `AStartDateTime` and `AEndDateTime`, and returns `True` if it is. When `AInclusive` is `True` (the default), then the limits are included. When `AInclusive` is `False`, the limits are excluded.

Errors: The `AStartDateTime` value must be before `AEndDateTime`, but no check is performed.

See also: DateInRange (604), TimeInRange (670)

44.4.7 DateTimeToDosDateTime

Synopsis: Convert `TDateTime` format to DOS date/time format.

Declaration:
```
function DateTimeToDosDateTime(const AValue: TDateTime) : LongInt
```

Visibility: default

Description: `DateTimeToDosDatetime` takes `Value`, a `TDateTime` formatted timestamp, and recodes it to a MS-DOS encoded date/time value. This is a longint with the date/time encoded in the bits as:

0-4 Seconds divided by 2

5-10 Minutes

11-15 Hours

16-20 Day

21-24 Month

25-31 Years since 1980

See also: DosDateTimeToDateTime (615)

44.4.8 DateTimeToJulianDate

Synopsis: Converts a TDateTime value to a Julian date representation.

Declaration: `function DateTimeToJulianDate(const AValue: TDateTime) : Double`

Visibility: default

Description: `DateTimeToJulianDate` converts the `AValue` date/time indication to a Julian (as opposed to Gregorian) date.

See also: JulianDateToDateTime (639), TryJulianDateToDateTime (678), DateTimeToModifiedJulianDate (606), TryModifiedJulianDateToDateTime (678)

44.4.9 DateTimeToMac

Synopsis: Convert a `TDateTime` timestamp to a Mac timestamp.

Declaration: `function DateTimeToMac(const AValue: TDateTime) : Int64`

Visibility: default

Description: `DateTimeToMac` converts the `TDateTime` value `AValue` to a valid Mac timestamp indication and returns the result.

Errors: None.

See also: UnixTimeStampToMac (680), MacToDateTime (639), MacTimeStampToUnix (639)

44.4.10 DateTimeToModifiedJulianDate

Synopsis: Convert a `TDateTime` value to a modified Julian date representation.

Declaration: `function DateTimeToModifiedJulianDate(const AValue: TDateTime) : Double`

Visibility: default

Description: Not yet implemented.

Errors: Currently, trying to use this function will raise an exception.

See also: DateTimeToJulianDate (606), JulianDateToDateTime (639), TryJulianDateToDateTime (678), TryModifiedJulianDateToDateTime (678)

44.4.11 DateTimeToUnix

Synopsis: Convert a `TDateTime` value to Unix epoch time.

Declaration:
```
function DateTimeToUnix(const AValue: TDateTime; AInputIsUTC: Boolean)
                        : Int64
```

Visibility: default

Description: `DateTimeToUnix` converts a `TDateTime` value to a epoch time (i.e. the number of seconds elapsed since 1/1/1970).

See also: UnixToDateTime (680)

44.4.12 DateToISO8601

Synopsis: Converts a TDateTime value to ISO 8601 date/time format.

Declaration:
```
function DateToISO8601(const ADate: TDateTime; AInputIsUTC: Boolean)
                                : string
```

Visibility: default

Description: `DateToISO8601` is a `String` function used to convert the `TDateTime` value in ADate to ISO 8601 date/time notation.

`ADate` contains the native `TDateTime` value converted in the function.

`AInputisUTC` indicates if the value in `ADate` represents a date/time value for the UTC time zone. When `AInputIsUTC` contains True, the 'Z' (Zulu time) time zone designation is used in the converted ISO 8601 value. Otherwise, the time zone is expressed as a positive or negative number of hours and minutes (such as "-04:00") in the return value.

`DateToISO8601` calls `GetLocalTimeOffset` to determine the time zone offset in use on the local computer. The integer offset is used to adjust the value in ADate to the UTC time zone when necessary.

The return value contains the adjusted value in ADate formatted using the notation:

- yyyy-mm-ddThh:nn:ss.zzz±hh:nn or
- yyyy-mm-ddThh:nn:ss.zzzZ for Zulu time

Use `ISO8601ToDate` to convert the return value back to a native `TDateTime` type.

See also: ISO8601ToDate (632)

44.4.13 DayOf

Synopsis: Extract the day (of month) part from a TDateTime value.

Declaration:
```
function DayOf(const AValue: TDateTime) : Word
```

Visibility: default

Description: `DayOf` returns the day of the month part of the `AValue` date/time indication. It is a number between 1 and 31.

For an example, see YearOf (693)

See also: YearOf (693), WeekOf (680), MonthOf (648), HourOf (622), MinuteOf (644), SecondOf (661), MilliSecondOf (640)

44.4.14 DayOfTheMonth

Synopsis: Extract the day (of month) part of a TDateTime value.

Declaration: `function DayOfTheMonth(const AValue: TDateTime) : Word`

Visibility: default

Description: `DayOfTheMonth` returns the number of days that have passed since the start of the month till the moment indicated by `AValue`. This is a one-based number, i.e. the first day of the month will return 1.

For an example, see the WeekOfTheMonth (681) function.

See also: DayOfTheYear (608), WeekOfTheMonth (681), HourOfTheMonth (623), MinuteOfTheMonth (645), SecondOfTheMonth (663), MilliSecondOfTheMonth (641)

44.4.15 DayOfTheWeek

Synopsis: Extracts the day of the week from a TDateTime value.

Declaration: `function DayOfTheWeek(const AValue: TDateTime) : Word`

Visibility: default

Description: `DayOfTheWeek` returns the number of days that have passed since the start of the week till the moment indicated by `AValue`. This is a one-based number, i.e. the first day of the week will return 1.

See also: DayOfTheYear (608), DayOfTheMonth (608), HourOfTheWeek (623), MinuteOfTheWeek (645), SecondOfTheWeek (663), MilliSecondOfTheWeek (642)

Listing: ./examples/datutex/ex42.pp

```
Program Example42;

{ This program demonstrates the WeekOfTheMonth function }

Uses SysUtils, DateUtils;

Var
  N : TDateTime;

Begin
  N:=Now;
  Writeln('Day of the Week          : ',DayOfTheWeek(N));
  Writeln('Hour of the Week         : ',HourOfTheWeek(N));
  Writeln('Minute of the Week       : ',MinuteOfTheWeek(N));
  Writeln('Second of the Week       : ',SecondOfTheWeek(N));
  Writeln('MilliSecond of the Week : ',
          MilliSecondOfTheWeek(N));
End.
```

44.4.16 DayOfTheYear

Synopsis: Extracts the day of the year from a `TDateTime` value.

Declaration: `function DayOfTheYear(const AValue: TDateTime) : Word`

Visibility: default

Description: `DayOfTheYear` returns the number of days that have passed since the start of the year till the moment indicated by `AValue`. This is a one-based number, i.e. January 1 will return 1.

For an example, see the WeekOfTheYear (681) function.

See also: WeekOfTheYear (681), HourOfTheYear (624), MinuteOfTheYear (646), SecondOfTheYear (663), MilliSecondOfTheYear (642)

44.4.17 DaysBetween

Synopsis: Number of whole days between two TDateTime values.

Declaration:
```
function DaysBetween(const ANow: TDateTime; const AThen: TDateTime)
                          : Integer
```

Visibility: default

Description: `DaysBetween` returns the number of whole days between `ANow` and `AThen`. This means the fractional part of a day (hours, minutes, etc.) is dropped.

See also: YearsBetween (693), MonthsBetween (649), WeeksBetween (682), HoursBetween (624), MinutesBetween (646), SecondsBetween (664), MilliSecondsBetween (642)

Listing: ./examples/datutex/ex58.pp

```
Program Example58;

{ This program demonstrates the DaysBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of days between ');
  Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));
  Writeln(' : ',DaysBetween(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=Today-23/24;
  Test(D1,D2);
  D2:=Today-1;
  Test(D1,D2);
  D2:=Today-25/24;
  Test(D1,D2);
  D2:=Today-26/24;
  Test(D1,D2);
  D2:=Today-5.4;
  Test(D1,D2);
  D2:=Today-2.5;
  Test(D1,D2);
End.
```

44.4.18 DaysInAMonth

Synopsis: Number of days in a month of a certain year.

Declaration: `function DaysInAMonth(const AYear: Word; const AMonth: Word) : Word`

Visibility: default

Description: `DaysInYMonth` returns the number of days in the month `AMonth` in the year`AYear`. The return value takes leap years into account.

See also: WeeksInAYear (683), WeeksInYear (684), DaysInYear (611), DaysInAYear (610), DaysInMonth (611)

Listing: ./examples/datutex/ex17.pp

```
Program Example17;

{ This program demonstrates the DaysInAMonth function }

Uses SysUtils, DateUtils;

Var
  Y,M : Word;

Begin
  For Y:=1992 to 2010 do
    For M:=1 to 12 do
      Writeln(LongMonthNames[m],' ',Y,' has ',DaysInAMonth(Y,M),' days.');
End.
```

44.4.19 DaysInAYear

Synopsis: Number of days in a particular year.

Declaration: `function DaysInAYear(const AYear: Word) : Word`

Visibility: default

Description: `DaysInAYear` returns the number of weeks in the year `AYear`. The return value is either 365 or 366.

See also: WeeksInAYear (683), WeeksInYear (684), DaysInYear (611), DaysInMonth (611), DaysInAMonth (610)

Listing: ./examples/datutex/ex15.pp

```
Program Example15;

{ This program demonstrates the DaysInAYear function }

Uses SysUtils, DateUtils;

Var
  Y : Word;

Begin
  For Y:=1992 to 2010 do
    Writeln(Y,' has ',DaysInAYear(Y),' days.');
End.
```

44.4.20 DaysInMonth

Synopsis: Return the number of days in the month in which a date occurs.

Declaration: `function DaysInMonth(const AValue: TDateTime) : Word`

Visibility: default

Description: `DaysInMonth` returns the number of days in the month in which `AValue` falls. The return value takes leap years into account.

See also: WeeksInAYear (683), WeeksInYear (684), DaysInYear (611), DaysInAYear (610), DaysInAMonth (610)

Listing: ./examples/datutex/ex16.pp

```
Program Example16;

{ This program demonstrates the DaysInMonth function }

Uses SysUtils, DateUtils;

Var
  Y,M : Word;

Begin
  For Y:=1992 to 2010 do
    For M:=1 to 12 do
      WriteIn(LongMonthNames[m],' ',Y,' has ',DaysInMonth(EncodeDate(Y,M,1)),' days.');
End.
```

44.4.21 DaysInYear

Synopsis: Return the number of days in the year in which a date occurs.

Declaration: `function DaysInYear(const AValue: TDateTime) : Word`

Visibility: default

Description: `daysInYear` returns the number of days in the year part of `AValue`. The return value is either 365 or 366.

See also: WeeksInAYear (683), WeeksInYear (684), DaysInAYear (610), DaysInMonth (611), DaysInAMonth (610)

Listing: ./examples/datutex/ex14.pp

```
Program Example14;

{ This program demonstrates the DaysInYear function }

Uses SysUtils, DateUtils;

Var
  Y : Word;

Begin
  For Y:=1992 to 2010 do
    WriteIn(Y,' has ',DaysInYear(EncodeDate(Y,1,1)),' days.');
End.
```

44.4.22 DaySpan

Synopsis: Calculate the approximate number of days between two TDateTime values.

Declaration: `function DaySpan(const ANow: TDateTime; const AThen: TDateTime) : Double`

Visibility: default

Description: `DaySpan` returns the number of Days between `ANow` and `AThen`, including any fractional parts of a Day.

See also: YearSpan (694), MonthSpan (649), WeekSpan (684), HourSpan (625), MinuteSpan (647), SecondSpan (664), MilliSecondSpan (643), DaysBetween (609)

Listing: ./examples/datutex/ex66.pp

```
Program Example66;

{ This program demonstrates the DaySpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of days between ');
  Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));
  Writeln(' : ',DaySpan(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=Today-23/24;
  Test(D1,D2);
  D2:=Today-1;
  Test(D1,D2);
  D2:=Today-25/24;
  Test(D1,D2);
  D2:=Today-26/24;
  Test(D1,D2);
  D2:=Today-5.4;
  Test(D1,D2);
  D2:=Today-2.5;
  Test(D1,D2);
End.
```

44.4.23 DecodeDateDay

Synopsis: Decode a TDateTime value in year and year of day.

Declaration: `procedure DecodeDateDay(const AValue: TDateTime; out AYear: Word; out ADayOfYear: Word)`

Visibility: default

Description: `DecodeDateDay` decomposes the date indication in `AValue` and returns the various components in `AYear`, `ADayOfYear`.

See also: EncodeDateTime (616), EncodeDateMonthWeek (616), EncodeDateWeek (617), EncodeDateDay (616), DecodeDateTime (614), DecodeDateWeek (614), DecodeDateMonthWeek (613)

Listing: ./examples/datutex/ex83.pp

```
Program Example83;

{ This program demonstrates the DecodeDateDay function }

Uses SysUtils, DateUtils;

Var
  Y,DoY : Word;
  TS : TDateTime;

Begin
  DecodeDateDay(Now,Y,DoY);
  TS:=EncodeDateDay(Y,DoY);
  Writeln('Today is : ',DateToStr(TS));
End.
```

44.4.24 DecodeDateMonthWeek

Synopsis: Decode a TDateTime value in a month, week of month and day of week.

Declaration:
```
procedure DecodeDateMonthWeek(const AValue: TDateTime; out AYear: Word;
                              out AMonth: Word; out AWeekOfMonth: Word;
                              out ADayOfWeek: Word)
```

Visibility: default

Description: `DecodeDateMonthWeek` decomposes the date indication in `AValue` and returns the various components in `AYear`, `AMonthAWeekOfMonth` and `ADayOfWeek`.

See also: EncodeDateTime (616), EncodeDateMonthWeek (616), EncodeDateWeek (617), EncodeDateDay (616), DecodeDateTime (614), DecodeDateWeek (614), DecodeDateDay (612)

Listing: ./examples/datutex/ex85.pp

```
Program Example85;

{ This program demonstrates the DecodeDateMonthWeek function }

Uses SysUtils, DateUtils;

Var
  Y,M,Wom,Dow : Word;
  TS : TDateTime;

Begin
  DecodeDateMonthWeek(Now,Y,M,WoM,DoW);
  TS:=EncodeDateMonthWeek(Y,M,WoM,Dow);
  Writeln('Today is : ',DateToStr(TS));
End.
```

44.4.25 DecodeDateTime

Synopsis: Decode a TDateTime value in a date and time value.

Declaration:
```
procedure DecodeDateTime(const AValue: TDateTime; out AYear: Word;
                         out AMonth: Word; out ADay: Word;
                         out AHour: Word; out AMinute: Word;
                         out ASecond: Word; out AMilliSecond: Word)
```

Visibility: default

Description: `DecodeDateTime` decomposes the date/time indication in `AValue` and returns the various components in `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecond`, `AMilliSecond`

See also: EncodeDateTime (616), EncodeDateMonthWeek (616), EncodeDateWeek (617), EncodeDateDay (616), DecodeDateWeek (614), DecodeDateDay (612), DecodeDateMonthWeek (613)

Listing: ./examples/datutex/ex79.pp

```
Program Example79;

{ This program demonstrates the DecodeDateTime function }

Uses SysUtils, DateUtils;

Var
  Y,Mo,D,H,Mi,S,MS : Word;
  TS : TDateTime;

Begin
  DecodeDateTime(Now,Y,Mo,D,H,Mi,S,MS);
  TS:=EncodeDateTime(Y,Mo,D,H,Mi,S,MS);
  Writeln('Now is : ',DateTimeToStr(TS));
End.
```

44.4.26 DecodeDateWeek

Synopsis: Decode a TDateTime value in a week of year and day of week.

Declaration:
```
procedure DecodeDateWeek(const AValue: TDateTime; out AYear: Word;
                         out AWeekOfYear: Word; out ADayOfWeek: Word)
```

Visibility: default

Description: `DecodeDateWeek` decomposes the date indication in `AValue` and returns the various components in `AYear`, `AWeekOfYear`, `ADayOfWeek`.

See also: EncodeDateTime (616), EncodeDateMonthWeek (616), EncodeDateWeek (617), EncodeDateDay (616), DecodeDateTime (614), DecodeDateDay (612), DecodeDateMonthWeek (613)

Listing: ./examples/datutex/ex81.pp

```
Program Example81;

{ This program demonstrates the DecodeDateWeek function }

Uses SysUtils, DateUtils;
```

```
Var
  Y,W,Dow : Word;
  TS : TDateTime;

Begin
  DecodeDateWeek(Now,Y,W,Dow);
  TS:=EncodeDateWeek(Y,W,Dow);
  Writeln('Today is : ',DateToStr(TS));
End.
```

44.4.27 DecodeDayOfWeekInMonth

Synopsis: Decode a TDateTime value in year, month, day of week parts.

Declaration:
```
procedure DecodeDayOfWeekInMonth(const AValue: TDateTime;
                                 out AYear: Word; out AMonth: Word;
                                 out ANthDayOfWeek: Word;
                                 out ADayOfWeek: Word)
```

Visibility: default

Description: `DecodeDayOfWeekInMonth` decodes the date `AValue` in a `AYear`, `AMonth`, `ADayOfweek` and `ANthDayOfweek`. (This is the N-th time that this weekday occurs in the month, e.g. the third Saturday of the month.)

See also: NthDayOfWeek (650), EncodeDateMonthWeek (616), #rtl.sysutils.DayOfWeek (1699), EncodeDayOfWeekInMonth (617), TryEncodeDayOfWeekInMonth (674)

Listing: ./examples/datutex/ex105.pp

```
Program Example105;

{ This program demonstrates the DecodeDayOfWeekInMonth function }

Uses SysUtils,DateUtils;

Var
  Y,M,NDoW,DoW : Word;
  D : TDateTime;
Begin
  DecodeDayOfWeekInMonth(Date,Y,M,NDoW,DoW);
  D:=EncodeDayOfWeekInMonth(Y,M,NDoW,DoW);
  Write(DateToStr(D),' is the ',NDow,'-th ');
  Writeln(formatdateTime('dddd',D),' of the month.');
End.
```

44.4.28 DosDateTimeToDateTime

Synopsis: Convert DOS date/time format to `TDateTime` format.

Declaration:
```
function DosDateTimeToDateTime(AValue: LongInt) : TDateTime
```

Visibility: default

Description: `DosDateTimeToDateTime` takes a DOS encoded date/time `AValue` and recodes it as a `TDateTime` value.

The bit encoding of the DOS date/time is explained in the DateTimeToDosDateTime (605) function.

See also: DateTimeToDosDateTime (605)

44.4.29 EncodeDateDay

Synopsis: Encodes a year and day of year to a TDateTime value.

Declaration:
```
function EncodeDateDay(const AYear: Word; const ADayOfYear: Word)
                       : TDateTime
```

Visibility: default

Description: `EncodeDateDay` encodes the values `AYear` and `ADayOfYear` to a date value and returns this value.

For an example, see DecodeDateDay (612).

Errors: If any of the arguments is not valid, then an `EConvertError` exception is raised.

See also: EncodeDateMonthWeek (616), DecodeDateDay (612), EncodeDateTime (616), EncodeDateWeek (617), TryEncodeDateTime (673), TryEncodeDateMonthWeek (672), TryEncodeDateWeek (674)

44.4.30 EncodeDateMonthWeek

Synopsis: Encodes a year, month, week of month and day of week to a TDateTime value.

Declaration:
```
function EncodeDateMonthWeek(const AYear: Word; const AMonth: Word;
                             const AWeekOfMonth: Word;
                             const ADayOfWeek: Word) : TDateTime
```

Visibility: default

Description: `EncodeDateTime` encodes the values `AYearAMonth`, `WeekOfMonth`,`ADayOfWeek`, to a date value and returns this value.

For an example, see DecodeDateMonthWeek (613).

Errors: If any of the arguments is not valid, then an `EConvertError` exception is raised.

See also: DecodeDateMonthWeek (613), EncodeDateTime (616), EncodeDateWeek (617), EncodeDateDay (616), TryEncodeDateTime (673), TryEncodeDateWeek (674), TryEncodeDateMonthWeek (672), TryEncodeDateDay (671), NthDayOfWeek (650)

44.4.31 EncodeDateTime

Synopsis: Encodes a TDateTime value from all its parts.

Declaration:
```
function EncodeDateTime(const AYear: Word; const AMonth: Word;
                        const ADay: Word; const AHour: Word;
                        const AMinute: Word; const ASecond: Word;
                        const AMilliSecond: Word) : TDateTime
```

Visibility: default

Description: `EncodeDateTime` encodes the values `AYearAMonth`, `ADay`,`AHour`, `AMinute`,`ASecond` and `AMilliSecond` to a date/time value and returns this value.

For an example, see DecodeDateTime (614).

Errors: If any of the arguments is not valid, then an `EConvertError` exception is raised.

See also: DecodeDateTime (614), EncodeDateMonthWeek (616), EncodeDateWeek (617), EncodeDateDay (616), TryEncodeDateTime (673), TryEncodeDateWeek (674), TryEncodeDateDay (671), TryEncodeDateMonthWeek (672)

44.4.32 EncodeDateWeek

Synopsis: Encode a `TDateTime` value from a year, week and day of week triplet.

Declaration:
```
function EncodeDateWeek(const AYear: Word; const AWeekOfYear: Word;
                        const ADayOfWeek: Word) : TDateTime
function EncodeDateWeek(const AYear: Word; const AWeekOfYear: Word)
                        : TDateTime
```

Visibility: default

Description: `EncodeDateWeek` encodes the values `AYear`, `AWeekOfYear` and `ADayOfWeek` to a date value and returns this value.

For an example, see DecodeDateWeek (614).

Errors: If any of the arguments is not valid, then an `EConvertError` exception is raised.

See also: EncodeDateMonthWeek (616), DecodeDateWeek (614), EncodeDateTime (616), EncodeDateDay (616), TryEncodeDateTime (673), TryEncodeDateWeek (674), TryEncodeDateMonthWeek (672)

44.4.33 EncodeDayOfWeekInMonth

Synopsis: Encodes a year, month, week, day of week specification to a `TDateTime` value.

Declaration:
```
function EncodeDayOfWeekInMonth(const AYear: Word; const AMonth: Word;
                                const ANthDayOfWeek: Word;
                                const ADayOfWeek: Word) : TDateTime
```

Visibility: default

Description: `EncodeDayOfWeekInMonth` encodes `AYear`, `AMonth`, `ADayOfweek` and `ANthDayOfweek` to a valid date stamp and returns the result.

`ANthDayOfweek`is the N-th time that this weekday occurs in the month, e.g. the third Saturday of the month.

For an example, see DecodeDayOfWeekInMonth (615).

Errors: If any of the values is not in range, then an `EConvertError` exception will be raised.

See also: NthDayOfWeek (650), EncodeDateMonthWeek (616), #rtl.sysutils.DayOfWeek (1699), DecodeDayOfWeekInMonth (615), TryEncodeDayOfWeekInMonth (674)

44.4.34 EncodeTimeInterval

Synopsis: Encode an interval as a `TDateTime` value.

Declaration:
```
function EncodeTimeInterval(Hour: Word; Minute: Word; Second: Word;
                            MilliSecond: Word) : TDateTime
```

Visibility: default

Description: `EncodeTimeInterval` encodes a time interval expressed in `Hour`, `Min`, `Sec`, `MSec` as a `TDateTime` value and returns the value in `Time`.

Errors: If `Min`, `Sec`, `MSec` do not contain a valid time indication, then an `EConvertError` exception is raised.

See also: TryEncodeTimeInterval (675)

44.4.35 EndOfADay

Synopsis: Calculates a TDateTime value representing the end of a specified day.

Declaration:
```
function EndOfADay(const AYear: Word; const AMonth: Word;
                   const ADay: Word) : TDateTime;  Overload
function EndOfADay(const AYear: Word; const ADayOfYear: Word)
                    : TDateTime;  Overload
```

Visibility: default

Description: `EndOfADay` returns a `TDateTime` value with the date/time indication of the last moment (23:59:59.999) of the day given by `AYear`, `AMonth`, `ADay`.

The day may also be indicated with a `AYear`, `ADayOfYear` pair.

See also: StartOfTheDay (668), StartOfADay (665), StartOfTheWeek (669), StartOfAWeek (667), StartOfAMonth (666), StartOfTheMonth (668), EndOfTheWeek (621), EndOfAWeek (619), EndOfTheYear (622), EndOfAYear (620), EndOfTheMonth (621), EndOfAMonth (618), EndOfTheDay (620)

Listing: ./examples/datutex/ex39.pp

```
Program Example39;

{ This program demonstrates the EndOfADay function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"End of the day : "dd mmmm yyyy hh:nn:ss';

Var
  Y,M,D : Word;

Begin
  Y:=YearOf(Today);
  M:=MonthOf(Today);
  D:=DayOf(Today);
  WriteLn(FormatDateTime(Fmt,EndOfADay(Y,M,D)));
  DecodeDateDay(Today,Y,D);
  WriteLn(FormatDateTime(Fmt,EndOfADay(Y,D)));
End.
```

44.4.36 EndOfAMonth

Synopsis: Calculate a TDateTime value representing the last day of the indicated month.

Declaration:
```
function EndOfAMonth(const AYear: Word; const AMonth: Word) : TDateTime
```

Visibility: default

Description: `EndOfAMonth` e returns a `TDateTime` value with the date of the last day of the month indicated by the `AYear`, `AMonth` pair.

See also: StartOfTheMonth (668), StartOfAMonth (666), EndOfTheMonth (621), EndOfTheYear (622), EndOfAYear (620), StartOfAWeek (667), StartOfTheWeek (669)

Listing: ./examples/datutex/ex31.pp

```
Program Example31;

{ This program demonstrates the EndOfAMonth function }

Uses SysUtils,DateUtils;

Const
  Fmt = '"Last day of this month : "dd mmmm yyyy';
Var
  Y,M : Word;

Begin
  Y:=YearOf(Today);
  M:=MonthOf(Today);
  WriteLn(FormatDateTime(Fmt,EndOfAMonth(Y,M)));
End.
```

44.4.37 EndOfAWeek

Synopsis: Return the last moment of day of the week, given a year and a week in the year.

Declaration:
```
function EndOfAWeek(const AYear: Word; const AWeekOfYear: Word;
                    const ADayOfWeek: Word) : TDateTime
function EndOfAWeek(const AYear: Word; const AWeekOfYear: Word)
                    : TDateTime
```

Visibility: default

Description: `EndOfAWeek` returns a `TDateTime` value with the date of the last moment (23:59:59:999) on the indicated day of the week indicated by the `AYear`, `AWeek`, `ADayOfWeek` values.

The default value for `ADayOfWeek` is 7.

See also: StartOfTheWeek (669), EndOfTheWeek (621), EndOfAWeek (619), StartOfAMonth (666), EndOfTheYear (622), EndOfAYear (620), EndOfTheMonth (621), EndOfAMonth (618)

Listing: ./examples/datutex/ex35.pp

```
Program Example35;

{ This program demonstrates the EndOfAWeek function }

Uses SysUtils,DateUtils;

Const
  Fmt = '"Last day of this week : "dd mmmm yyyy hh:nn:ss';
  Fmt2 = '"Last-1 day of this week : "dd mmmm yyyy hh:nn:ss';

Var
  Y,W : Word;

Begin
  Y:=YearOf(Today);
```

```
  W:=WeekOf(Today);
  WriteLn(FormatDateTime(Fmt,EndOfAWeek(Y,W)));
  WriteLn(FormatDateTime(Fmt2,EndOfAWeek(Y,W,6)));
End.
```

44.4.38 EndOfAYear

Synopsis: Calculate a TDateTime value representing the last day of a year.

Declaration: `function EndOfAYear(const AYear: Word) : TDateTime`

Visibility: default

Description: `StartOfAYear` returns a `TDateTime` value with the date of the last day of the year `AYear` (December 31).

See also: StartOfTheYear (669), EndOfTheYear (622), EndOfAYear (620), EndOfTheMonth (621), EndOfAMonth (618), StartOfAWeek (667), StartOfTheWeek (669)

Listing: ./examples/datutex/ex27.pp

```
Program Example27;

{ This program demonstrates the EndOfAYear function }

Uses SysUtils,DateUtils;

Const
  Fmt = '"Last day of this year : "dd mmmm yyyy';

Begin
  WriteLn(FormatDateTime(Fmt,EndOfAYear(YearOf(Today))));
End.
```

44.4.39 EndOfTheDay

Synopsis: Calculate a TDateTime value that represents the end of a given day.

Declaration: `function EndOfTheDay(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `EndOfTheDay` extracts the date part of `AValue` and returns a `TDateTime` value with the date/time indication of the last moment (23:59:59.999) of this day.

See also: StartOftheDay (668), StartOfADay (665), StartOfTheWeek (669), StartOfAWeek (667), StartOfAMonth (666), StartOfTheMonth (668), EndOfTheWeek (621), EndOfAWeek (619), EndOfTheYear (622), EndOfAYear (620), EndOfTheMonth (621), EndOfAMonth (618), EndOfADay (618)

Listing: ./examples/datutex/ex37.pp

```
Program Example37;

{ This program demonstrates the EndOfTheDay function }

Uses SysUtils,DateUtils;
```

```
Const
  Fmt = '"End of the day : "dd mmmm yyyy hh:nn:ss';

Begin
  WriteLn(FormatDateTime(Fmt,EndOfTheDay(Today)));
End.
```

44.4.40 EndOfTheMonth

Synopsis: Calculate a TDateTime value representing the last day of the month, given a day in that month.

Declaration: `function EndOfTheMonth(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `EndOfTheMonth` extracts the year and month parts of `AValue` and returns a `TDateTime` value with the date of the first day of that year and month as the EndOfAMonth (618) function.

See also: StartOfAMonth (666), StartOfTheMonth (668), EndOfAMonth (618), EndOfTheYear (622), EndOfAYear (620), StartOfAWeek (667), StartOfTheWeek (669)

Listing: ./examples/datutex/ex29.pp

```
Program Example29;

{ This program demonstrates the EndOfTheMonth function }

Uses SysUtils,DateUtils;

Const
  Fmt = '"last day of this month : "dd mmmm yyyy';

Begin
  WriteLn(FormatDateTime(Fmt,EndOfTheMonth(Today)));
End.
```

44.4.41 EndOfTheWeek

Synopsis: Calculate a TDateTime value which represents the end of a week, given a date in that week.

Declaration: `function EndOfTheWeek(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `EndOfTheWeek` extracts the year and week parts of `AValue` and returns a `TDateTime` value with the date of the last day of that week as the EndOfAWeek (619) function.

See also: StartOfAWeek (667), StartOfTheWeek (669), EndOfAWeek (619), StartOfAMonth (666), EndOfTheYear (622), EndOfAYear (620), EndOfTheMonth (621), EndOfAMonth (618)

Listing: ./examples/datutex/ex33.pp

```
Program Example33;

{ This program demonstrates the EndOfTheWeek function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"last day of this week : "dd mmmm yyyy';

Begin
  WriteLn(FormatDateTime(Fmt, EndOfTheWeek(Today)));
End.
```

44.4.42 EndOfTheYear

Synopsis: Calculate a TDateTime value representing the last day of a year, given a date in that year.

Declaration: `function EndOfTheYear(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `EndOfTheYear` extracts the year part of `AValue` and returns a `TDateTime` value with the date of the last day of that year (December 31), as the EndOfAYear (620) function.

See also: StartOfAYear (667), StartOfTheYear (669), EndOfTheMonth (621), EndOfAMonth (618), StartOfAWeek (667), StartOfTheWeek (669), EndOfAYear (620)

Listing: ./examples/datutex/ex25.pp

```
Program Example25;

{ This program demonstrates the EndOfTheYear function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"Last day of this year : "dd mmmm yyyy';

Begin
  WriteLn(FormatDateTime(Fmt, EndOfTheYear(Today)));
End.
```

44.4.43 HourOf

Synopsis: Extract the hour part from a TDateTime value.

Declaration: `function HourOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `HourOf` returns the hour of the day part of the `AValue` date/time indication. It is a number between 0 and 23.

For an example, see YearOf (693)

See also: YearOf (693), WeekOf (680), MonthOf (648), DayOf (607), MinuteOf (644), SecondOf (661), MilliSecondOf (640)

44.4.44 HourOfTheDay

Synopsis: Calculate the hour of a given TDateTime value.

Declaration: `function HourOfTheDay(const AValue: TDateTime) : Word`

Visibility: default

Description: `HourOfTheDay` returns the number of hours that have passed since the start of the day till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:59:59 will return 0.

See also: HourOfTheYear (624), HourOfTheMonth (623), HourOfTheWeek (623), MinuteOfTheDay (644), SecondOfTheDay (662), MilliSecondOfTheDay (640)

Listing: ./examples/datutex/ex43.pp

```
Program Example43;

{ This program demonstrates the HourOfTheDay function }

Uses SysUtils, DateUtils;

Var
  N : TDateTime;

Begin
  N:=Now;
  Writeln('Hour of the Day        : ',HourOfTheDay(N));
  Writeln('Minute of the Day      : ',MinuteOfTheDay(N));
  Writeln('Second of the Day      : ',SecondOfTheDay(N));
  Writeln('MilliSecond of the Day : ',
          MilliSecondOfTheDay(N));
End.
```

44.4.45 HourOfTheMonth

Synopsis: Calculate the number of hours passed since the start of the month.

Declaration: `function HourOfTheMonth(const AValue: TDateTime) : Word`

Visibility: default

Description: `HourOfTheMonth` returns the number of hours that have passed since the start of the month till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:59:59 on the first day of the month will return 0.

For an example, see the WeekOfTheMonth (681) function.

See also: WeekOfTheMonth (681), DayOfTheMonth (608), MinuteOfTheMonth (645), SecondOfTheMonth (663), MilliSecondOfTheMonth (641)

44.4.46 HourOfTheWeek

Synopsis: Calculate the number of hours elapsed since the start of the week.

Declaration: `function HourOfTheWeek(const AValue: TDateTime) : Word`

Visibility: default

Description: `HourOfTheWeek` returns the number of hours that have passed since the start of the Week till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:59:59 on the first day of the week will return 0.

For an example, see the DayOfTheWeek (608) function.

See also: HourOfTheYear (624), HourOfTheMonth (623), HourOfTheDay (623), DayOfTheWeek (608), MinuteOfTheWeek (645), SecondOfTheWeek (663), MilliSecondOfTheWeek (642)

44.4.47 HourOfTheYear

Synopsis: Calculate the number of hours passed since the start of the year.

Declaration: `function HourOfTheYear(const AValue: TDateTime) : Word`

Visibility: default

Description: `HourOfTheYear` returns the number of hours that have passed since the start of the year (January 1, 00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. January 1 00:59:59 will return 0.

For an example, see the WeekOfTheYear (681) function.

See also: WeekOfTheYear (681), DayOfTheYear (608), MinuteOfTheYear (646), SecondOfTheYear (663), MilliSecondOfTheYear (642)

44.4.48 HoursBetween

Synopsis: Calculate the number of whole hours between two TDateTime values.

Declaration: `function HoursBetween(const ANow: TDateTime; const AThen: TDateTime) : Int64`

Visibility: default

Description: `HoursBetween` returns the number of whole hours between `ANow` and `AThen`. This means the fractional part of an hour (minutes,seconds etc.) is dropped.

See also: YearsBetween (693), MonthsBetween (649), WeeksBetween (682), DaysBetween (609), MinutesBetween (646), SecondsBetween (664), MilliSecondsBetween (642)

Listing: ./examples/datutex/ex59.pp

```
Program Example59;

{ This program demonstrates the HoursBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of hours between ');
  Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));
  Writeln(' : ',HoursBetween(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;
```

```
Begin
  D1:=Now;
  D2:=D1-(59*OneMinute);
  Test(D1,D2);
  D2:=D1-(61*OneMinute);
  Test(D1,D2);
  D2:=D1-(122*OneMinute);
  Test(D1,D2);
  D2:=D1-(306*OneMinute);
  Test(D1,D2);
  D2:=D1-(5.4*OneHour);
  Test(D1,D2);
  D2:=D1-(2.5*OneHour);
  Test(D1,D2);
End.
```

44.4.49 HourSpan

Synopsis: Calculate the approximate number of hours between two TDateTime values.

Declaration:
```
function HourSpan(const ANow: TDateTime; const AThen: TDateTime)
                        : Double
```

Visibility: default

Description: `HourSpan` returns the number of Hours between `ANow` and `AThen`, including any fractional parts of a Hour.

See also: YearSpan (694), MonthSpan (649), WeekSpan (684), DaySpan (612), MinuteSpan (647), SecondSpan (664), MilliSecondSpan (643), HoursBetween (624)

Listing: ./examples/datutex/ex67.pp

```
Program Example67;

{ This program demonstrates the HourSpan function }

Uses SysUtils,DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
 Write('Number of hours between ');
 Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));
 Writeln(' : ',HourSpan(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(59*OneMinute);
  Test(D1,D2);
  D2:=D1-(61*OneMinute);
  Test(D1,D2);
```

```
  D2:=D1-(122*OneMinute);
  Test(D1,D2);
  D2:=D1-(306*OneMinute);
  Test(D1,D2);
  D2:=D1-(5.4*OneHour);
  Test(D1,D2);
  D2:=D1-(2.5*OneHour);
  Test(D1,D2);
End.
```

44.4.50 IncDay

Synopsis: Increase a TDateTime value with a number of days.

Declaration:
```
function IncDay(const AValue: TDateTime; const ANumberOfDays: Integer)
                  : TDateTime
function IncDay(const AValue: TDateTime) : TDateTime
```

Visibility: default

Description: `IncDay` adds `ANumberOfDays` days to `AValue` and returns the resulting date/time.`ANumberOfDays` can be positive or negative.

See also: IncYear (629), #rtl.sysutils.IncMonth (1756), IncWeek (628), IncHour (626), IncMinute (627), IncSecond (628), IncMilliSecond (627)

Listing: ./examples/datutex/ex74.pp

```
Program Example74;

{ This program demonstrates the IncDay function }

Uses SysUtils,DateUtils;

Begin
  WriteIn('One Day from today is ',DateToStr(IncDay(Today,1)));
  WriteIn('One Day ago from today is ',DateToStr(IncDay(Today,-1)));
End.
```

44.4.51 IncHour

Synopsis: Increase a TDateTime value with a number of hours.

Declaration:
```
function IncHour(const AValue: TDateTime; const ANumberOfHours: Int64)
                   : TDateTime
function IncHour(const AValue: TDateTime) : TDateTime
```

Visibility: default

Description: `IncHour` adds `ANumberOfHours` hours to `AValue` and returns the resulting date/time.`ANumberOfHours` can be positive or negative.

See also: IncYear (629), #rtl.sysutils.IncMonth (1756), IncWeek (628), IncDay (626), IncMinute (627), IncSecond (628), IncMilliSecond (627)

Listing: ./examples/datutex/ex75.pp

```
Program Example75
;

{ This program demonstrates the IncHour function }

Uses SysUtils, DateUtils;

Begin
  WriteLn('One Hour from now is ',DateTimeToStr(IncHour(Now,1)));
  WriteLn('One Hour ago from now is ',DateTimeToStr(IncHour(Now,-1)));
End.
```

44.4.52 IncMilliSecond

Synopsis: Increase a TDateTime value with a number of milliseconds.

Declaration:
```
function IncMilliSecond(const AValue: TDateTime;
                        const ANumberOfMilliSeconds: Int64) : TDateTime
function IncMilliSecond(const AValue: TDateTime) : TDateTime
```

Visibility: default

Description: `IncMilliSecond` adds `ANumberOfMilliSeconds` milliseconds to `AValue` and returns the resulting date/time.`ANumberOfMilliSeconds` can be positive or negative.

See also: IncYear (629), #rtl.sysutils.IncMonth (1756), IncWeek (628), IncDay (626), IncHour (626), IncSecond (628), IncMilliSecond (627)

Listing: ./examples/datutex/ex78.pp

```
Program Example78;

{ This program demonstrates the IncMilliSecond function }

Uses SysUtils, DateUtils;

Begin
  WriteLn('One MilliSecond from now is ',TimeToStr(IncMilliSecond(Now,1)));
  WriteLn('One MilliSecond ago from now is ',TimeToStr(IncMilliSecond(Now,-1)));
End.
```

44.4.53 IncMinute

Synopsis: Increase a TDateTime value with a number of minutes.

Declaration:
```
function IncMinute(const AValue: TDateTime;
                   const ANumberOfMinutes: Int64) : TDateTime
function IncMinute(const AValue: TDateTime) : TDateTime
```

Visibility: default

Description: `IncMinute` adds `ANumberOfMinutes` minutes to `AValue` and returns the resulting date/time.`ANumberOfMinutes` can be positive or negative.

See also: IncYear (629), #rtl.sysutils.IncMonth (1756), IncWeek (628), IncDay (626), IncHour (626), IncSecond (628), IncMilliSecond (627)

Listing: ./examples/datutex/ex76.pp

```
Program Example76;

{ This program demonstrates the IncMinute function }

Uses SysUtils, DateUtils;

Begin
  WriteLn('One Minute from now is ',TimeToStr(IncMinute(Time,1)));
  WriteLn('One Minute ago from now is ',TimeToStr(IncMinute(Time,-1)));
End.
```

44.4.54 IncSecond

Synopsis: Increase a TDateTime value with a number of seconds.

Declaration:
```
function IncSecond(const AValue: TDateTime;
                   const ANumberOfSeconds: Int64) : TDateTime
function IncSecond(const AValue: TDateTime) : TDateTime
```

Visibility: default

Description: `IncSecond` adds `ANumberOfSeconds` seconds to `AValue` and returns the resulting date/time.`ANumberOfSeconds` can be positive or negative.

See also: IncYear (629), #rtl.sysutils.IncMonth (1756), IncWeek (628), IncDay (626), IncHour (626), IncSecond (628), IncMilliSecond (627)

Listing: ./examples/datutex/ex77.pp

```
Program Example77;

{ This program demonstrates the IncSecond function }

Uses SysUtils, DateUtils;

Begin
  WriteLn('One Second from now is ',TimeToStr(IncSecond(Time,1)));
  WriteLn('One Second ago from now is ',TimeToStr(IncSecond(Time,-1)));
End.
```

44.4.55 IncWeek

Synopsis: Increase a TDateTime value with a number of weeks.

Declaration:
```
function IncWeek(const AValue: TDateTime; const ANumberOfWeeks: Integer)
                 : TDateTime
function IncWeek(const AValue: TDateTime) : TDateTime
```

Visibility: default

Description: `IncWeek` adds `ANumberOfWeeks` weeks to `AValue` and returns the resulting date/time.`ANumberOfWeeks` can be positive or negative.

See also: IncYear (629), #rtl.sysutils.IncMonth (1756), IncDay (626), IncHour (626), IncMinute (627), IncSecond (628), IncMilliSecond (627)

Listing: ./examples/datutex/ex73.pp

```
Program Example73;

{ This program demonstrates the IncWeek function }

Uses SysUtils,DateUtils;

Begin
  WriteLn('One Week from today is ',DateToStr(IncWeek(Today,1)));
  WriteLn('One Week ago from today is ',DateToStr(IncWeek(Today,-1)));
End.
```

44.4.56 IncYear

Synopsis: Increase a TDateTime value with a number of years.

Declaration:
```
function IncYear(const AValue: TDateTime; const ANumberOfYears: Integer)
                    : TDateTime
function IncYear(const AValue: TDateTime) : TDateTime
```

Visibility: default

Description: `IncYear` adds `ANumberOfYears` years to `AValue` and returns the resulting date/time. `ANumberOfYears` can be positive or negative.

See also: #rtl.sysutils.IncMonth (1756), IncWeek (628), IncDay (626), IncHour (626), IncMinute (627), IncSecond (628), IncMilliSecond (627)

Listing: ./examples/datutex/ex71.pp

```
Program Example71;

{ This program demonstrates the IncYear function }

Uses SysUtils,DateUtils;

Begin
  WriteLn('One year from today is ',DateToStr(IncYear(Today,1)));
  WriteLn('One year ago from today is ',DateToStr(IncYear(Today,-1)));
End.
```

44.4.57 InvalidDateDayError

Synopsis: Raise an `EConvertError` exception when a day is not a valid day of a year.

Declaration:
```
procedure InvalidDateDayError(const AYear: Word; const ADayOfYear: Word)
```

Visibility: default

Description: `InvalidDateDayError` raises an EConvertError (1830) exception and formats the error message with an appropriate description made up from the parts `AYear` and `ADayOfYear`.

Normally this function should not be needed, the conversion routines call it when they have received invalid arguments.

See also: InvalidDateWeekError (630), InvalidDateTimeError (630), InvalidDateMonthWeekError (630), InvalidDayOfWeekInMonthError (631)

44.4.58 InvalidDateMonthWeekError

Synopsis: Raise an EConvertError exception when a Year,Month,WeekOfMonth,DayofWeek is invalid.

Declaration:
```
procedure InvalidDateMonthWeekError(const AYear: Word;
                                    const AMonth: Word;
                                    const AWeekOfMonth: Word;
                                    const ADayOfWeek: Word)
```

Visibility: default

Description: InvalidDateMonthWeekError raises an EConvertError (1830) exception and formats the error message with an appropriate description made up from the parts AYear, Amonth, AWeekOfMonth and ADayOfWeek.

Normally this function should not be needed, the conversion routines call it when they have received invalid arguments.

See also: InvalidDateWeekError (630), InvalidDateTimeError (630), InvalidDateDayError (629), InvalidDayOfWeekInMonthError (631)

44.4.59 InvalidDateTimeError

Synopsis: Raise an EConvertError about an invalid date-time specification.

Declaration:
```
procedure InvalidDateTimeError(const AYear: Word; const AMonth: Word;
                               const ADay: Word; const AHour: Word;
                               const AMinute: Word; const ASecond: Word;
                               const AMilliSecond: Word;
                               const ABaseDate: TDateTime)
procedure InvalidDateTimeError(const AYear: Word; const AMonth: Word;
                               const ADay: Word; const AHour: Word;
                               const AMinute: Word; const ASecond: Word;
                               const AMilliSecond: Word)
```

Visibility: default

Description: InvalidDateTimeError raises an EConvertError (1830) exception and formats the error message with an appropriate description made up from the parts AYear, AMonth, ADay,AHour, AMinute, ASecond and AMilliSecond.

Normally this function should not be needed, the conversion routines call it when they have received invalid arguments.

See also: InvalidDateWeekError (630), InvalidDateDayError (629), InvalidDateMonthWeekError (630), InvalidDayOfWeekInMonthError (631)

44.4.60 InvalidDateWeekError

Synopsis: Raise an EConvertError with an invalid Year, WeekOfyear and DayOfWeek specification.

Declaration:
```
procedure InvalidDateWeekError(const AYear: Word;
                               const AWeekOfYear: Word;
                               const ADayOfWeek: Word)
```

Visibility: default

Description: `InvalidDateWeekError` raises an EConvertError (1830) exception and formats the error message with an appropriate description made up from the parts `AYear`, `AWeek`, `ADayOfWeek`

Normally this function should not be needed, the conversion routines call it when they have received invalid arguments.

See also: InvalidDateTimeError (630), InvalidDateDayError (629), InvalidDateMonthWeekError (630), InvalidDayOfWeekInMonthError (631)

44.4.61 InvalidDayOfWeekInMonthError

Synopsis: Raise an `EConvertError` exception when a Year,Month,NthDayofWeek,DayofWeek is invalid.

Declaration:
```
procedure InvalidDayOfWeekInMonthError(const AYear: Word;
                                       const AMonth: Word;
                                       const ANthDayOfWeek: Word;
                                       const ADayOfWeek: Word)
```

Visibility: default

Description: `InvalidDayOfWeekInMonthError` raises an EConvertError (1830) exception and formats the error message with an appropriate description made up from the parts `AYear`, `Amonth`, `ANthDayOfWeek` and `ADayOfWeek`.

Normally this function should not be needed, the conversion routines call it when they have received invalid arguments.

See also: InvalidDateWeekError (630), InvalidDateTimeError (630), InvalidDateDayError (629), InvalidDateMonthWeekError (630)

44.4.62 IsAM

Declaration:
```
function IsAM(const AValue: TDateTime) : Boolean
```

Visibility: default

44.4.63 IsInLeapYear

Synopsis: Determine whether a date is in a leap year.

Declaration:
```
function IsInLeapYear(const AValue: TDateTime) : Boolean
```

Visibility: default

Description: `IsInLeapYear` returns `True` if the year part of `AValue` is leap year, or `False` if not.

See also: YearOf (693), IsPM (633), IsToday (634), IsSameDay (633)

Listing: ./examples/datutex/ex3.pp

```
Program Example3;

{ This program demonstrates the IsInLeapYear function }

Uses SysUtils, DateUtils;

Begin
  WriteIn('Current year is leap year: ',IsInLeapYear(Date));
End.
```

44.4.64 ISO8601ToDate

Synopsis: Converts a date/time value in ISO 8601 notation to a native TDateTime type.

Declaration:
```
function ISO8601ToDate(const DateString: string; ReturnUTC: Boolean)
                       : TDateTime
```

Visibility: default

Description: `ISO8601ToDate` is a function used to convert a string using ISO 8601 date/time notation to a native `TDateTime` type.

`DateString` contains the date/time value expressed using ISO 8601 notation.

`ReturnUTC` indicates if the `TDateTime` value should be adjusted to reflect the UTC (Coordinated Universal Time) time zone. The default value for the argument is `True`.

`ISO8601ToDate` calls the `TryISO8601ToDate` function to perform the conversion, and raises an `EConvertError` exception if `DateString` contains an invalid ISO 8601 date/time value. The return value contains the native `TDateTime` value for the timestamp (adjusted to UTC when requested).

Use `DateToISO8601` to convert the native date/time value back to its representation using ISO 8601 date/time notation.

See also: TryISO8601ToDate (675), DateToISO8601 (607)

44.4.65 ISO8601ToDateDef

Synopsis: Converts an ISO 8601 date/time string to a TDateTime type.

Declaration:
```
function ISO8601ToDateDef(const DateString: string; ReturnUTC: Boolean;
                          aDefault: TDateTime) : TDateTime
function ISO8601ToDateDef(const DateString: string;
                          aDefault: TDateTime; ReturnUTC: Boolean)
                           : TDateTime
```

Visibility: default

Description: `ISO8601ToDateDef` is used to convert a date/time string in ISO 8601 format to its representation as a `TDateTime` value. DateString contains the ISO 8601-formatted date/time value converted in the routine. It can use one of the supported ISO 8601 date/time formats, and may contain an optional time zone offset. DateString uses the format returned from the `DateToISO8601` function. Each of the following represent the date/time for Noon on July 4, 2019:

- 20190714 12:00
- 2019-07-14 12:00
- 20190714 12:00:00
- 2019-07-14 12:00:00
- 20190714T12:00
- 20190714T12:00:00
- 2019-07-14T12:00
- 2019-07-14T12:00:00
- 20190714T12:00:00-04:00
- 2019-07-14T12:00:00-04:00

Time values in DateString specified without a time zone offset are assumed to be in the local time zone.

ReturnUTC indicates if the value in DateString is adjusted to UTC (Coordinated Universal Time) in the return value.

aDefault contains the default TDateTime value used as the return value when DateString cannot be successfully parsed and converted.

Please note than that the overloaded variant which uses String, Boolean, and TDateTime arguments has been deprecated. Use the variant with String, TDateTime, and Boolean arguments (in that order) instead.

ISO8601ToDateDef calls the TryISO8601ToDate function in its implementation.

See also: DateToISO8601 (607), TryISO8601ToDate (675)

44.4.66 IsPM

Synopsis: Determine whether a time is PM or AM.

Declaration: `function IsPM(const AValue: TDateTime) : Boolean`

Visibility: default

Description: `IsPM` returns `True` if the time part of `AValue` is later then 12:00 (PM, or afternoon).

See also: YearOf (693), IsInLeapYear (631), IsToday (634), IsSameDay (633)

Listing: ./examples/datutex/ex4.pp

```
Program Example4;

{ This program demonstrates the IsPM function }

Uses SysUtils, DateUtils;

Begin
  Writeln('Current time is PM : ',IsPM(Now));
End.
```

44.4.67 IsSameDay

Synopsis: Check if two date/time indications are the same day.

Declaration: `function IsSameDay(const AValue: TDateTime; const ABasis: TDateTime)
: Boolean`

Visibility: default

Description: `IsSameDay` checks whether `AValue` and `ABasis` have the same date part, and returns `True` if they do, `False` if not.

See also: Today (671), Yesterday (695), Tomorrow (671), IsToday (634)

Listing: ./examples/datutex/ex21.pp

```
Program Example21;

{ This program demonstrates the IsSameDay function }

Uses SysUtils, DateUtils;

Var
  I : Integer;
  D : TDateTime;

Begin
  For I:=1 to 3 do
    begin
    D:=Today+Random(3)-1;
    Write(FormatDateTime('dd mmmm yyyy "is today : "',D));
    WriteIn(IsSameDay(D,Today));
    end;
End.
```

44.4.68 IsSameMonth

Synopsis: Check if 2 dates are in the same month.

Declaration:
```
function IsSameMonth(const Avalue: TDateTime; const ABasis: TDateTime)
                          : Boolean
```

Visibility: default

Description: `IsSameMonth` will return `True` if the two dates `Avalue` and `ABasis` occur in the same year and month. (i.e. if their month and year parts match). Otherwise, `False` is returned.

See also: IsSameDay (633), IsToday (634), SameDate (658)

44.4.69 IsToday

Synopsis: Check whether a given date is today.

Declaration:
```
function IsToday(const AValue: TDateTime) : Boolean
```

Visibility: default

Description: `IsToday` returns `True` if `AValue` is today's date, and `False` otherwise.

See also: Today (671), Yesterday (695), Tomorrow (671), IsSameDay (633)

Listing: ./examples/datutex/ex20.pp

```
Program Example20;

{ This program demonstrates the IsToday function }

Uses SysUtils, DateUtils;

Begin
  WriteIn('Today     : ',IsToday(Today));
  WriteIn('Tomorrow  : ',IsToday(Tomorrow));
  WriteIn('Yesterday : ',IsToday(Yesterday));
End.
```

44.4.70 IsValidDate

Synopsis: Check whether a set of values is a valid date indication.

Declaration:
```
function IsValidDate(const AYear: Word; const AMonth: Word;
                     const ADay: Word) : Boolean
```

Visibility: default

Description: `IsValidDate` returns `True` when the values `AYear`, `AMonth`, `ADay` form a valid date indication. If one of the values is not valid (e.g. the day is invalid or does not exist in that particular month), `False` is returned.

`AYear` must be in the range 1..9999 to be valid.

See also: IsValidTime (638), IsValidDateTime (637), IsValidDateDay (635), IsValidDateWeek (638), IsValidDateMonthWeek (636)

Listing: ./examples/datutex/ex5.pp

```
Program Example5;

{ This program demonstrates the IsValidDate function }

Uses SysUtils, DateUtils;

Var
  Y,M,D : Word;

Begin
  For Y:=2000 to 2004 do
   For M:=1 to 12 do
     For D:=1 to 31 do
       If Not IsValidDate(Y,M,D) then
         Writeln(D,' is not a valid day in ',Y,'/',M);
End.
```

44.4.71 IsValidDateDay

Synopsis: Check whether a given year/day of year combination is a valid date.

Declaration:
```
function IsValidDateDay(const AYear: Word; const ADayOfYear: Word)
                        : Boolean
```

Visibility: default

Description: `IsValidDateDay` returns `True` if `AYear` and `ADayOfYear` form a valid date indication, or `False` otherwise.

`AYear` must be in the range 1..9999 to be valid.

The `ADayOfYear` value is checked to see whether it falls within the valid range of dates for `AYear`.

See also: IsValidDate (635), IsValidTime (638), IsValidDateTime (637), IsValidDateWeek (638), IsValidDateMonthWeek (636)

Listing: ./examples/datutex/ex9.pp

```
Program Example9;

{ This program demonstrates the IsValidDateDay function }

Uses SysUtils, DateUtils;

Var
  Y : Word;

Begin
  For Y:=1996 to 2004 do
    if IsValidDateDay(Y,366) then
      Writeln(Y,' is a leap year');
End.
```

44.4.72 IsValidDateMonthWeek

Synopsis: Check whether a given year/month/week/day of the week combination is a valid day.

Declaration:
```
function IsValidDateMonthWeek(const AYear: Word; const AMonth: Word;
                              const AWeekOfMonth: Word;
                              const ADayOfWeek: Word) : Boolean
```

Visibility: default

Description: `IsValidDateMonthWeek` returns `True` if `AYear`, `AMonthAWeekOfMonth` and `ADayOfWeek` form a valid date indication, or `False` otherwise.

`AYear` must be in the range 1..9999 to be valid.

The `AWeekOfMonth`,`ADayOfWeek` values are checked to see whether the combination falls within the valid range of weeks for the `AYear`,`AMonth` combination.

See also: IsValidDate (635), IsValidTime (638), IsValidDateTime (637), IsValidDateDay (635), IsValidDateWeek (638)

Listing: ./examples/datutex/ex11.pp

```
Program Example11;

{ This program demonstrates the IsValidDateMonthWeek function }

Uses SysUtils, DateUtils;

Var
  Y,W,D : Word;
  B : Boolean;

Begin
  For Y:=2000 to 2004 do
    begin
    B:=True;
    For W:=4 to 6 do
      For D:=1 to 7 do
        If B then
          begin
          B:=IsValidDateMonthWeek(Y,12,W,D);
          If Not B then
```

```
            if (D=1) then
              Writeln('December ',Y,' has exactly ',W,' weeks.')
            else
              Writeln('December ',Y,' has ',W,' weeks and ',D-1,' days.');
          end;
    end;
End.
```

44.4.73 IsValidDateTime

Synopsis: Check whether a set of values is a valid date and time indication.

Declaration:
```
function IsValidDateTime(const AYear: Word; const AMonth: Word;
                         const ADay: Word; const AHour: Word;
                         const AMinute: Word; const ASecond: Word;
                         const AMilliSecond: Word) : Boolean
```

Visibility: default

Description: `IsValidTime` returns `True` when the values `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecond` and `AMilliSecond` form a valid date and time indication. If one of the values is not valid (e.g. the seconds are larger than 60), `False` is returned.

`AYear` must be in the range 1..9999 to be valid.

See also: IsValidDate (635), IsValidTime (638), IsValidDateDay (635), IsValidDateWeek (638), IsValidDateMonthWeek (636)

Listing: ./examples/datutex/ex7.pp

```
Program Example7;

{ This program demonstrates the IsValidDateTime function }

Uses SysUtils,DateUtils;

Var
  Y,Mo,D : Word;
  H,M,S,MS : Word;
  I : Integer;

Begin
  For I:=1 to 10 do
    begin
    Y:=2000+Random(5);
    Mo:=Random(15);
    D:=Random(40);
    H:=Random(32);
    M:=Random(90);
    S:=Random(90);
    MS:=Random(1500);
    If Not IsValidDateTime(Y,Mo,D,H,M,S,MS) then
      Writeln(Y,'-',Mo,'-',D,' ',H,':',M,':',S,'.',MS,' is not a valid date/time.');
    end;
End.
```

44.4.74 IsValidDateWeek

Synopsis: Check whether a given year/week/day of the week combination is a valid day.

Declaration:
```
function IsValidDateWeek(const AYear: Word; const AWeekOfYear: Word;
                         const ADayOfWeek: Word) : Boolean
```

Visibility: default

Description: `IsValidDateWeek` returns `True` if `AYear`, `AWeekOfYear` and `ADayOfWeek` form a valid date indication, or `False` otherwise.

`AYear` must be in the range 1..9999 to be valid.

The `ADayOfWeek`,`ADayOfWeek` values are checked to see whether the combination falls within the valid range of weeks for `AYear`.

See also: IsValidDate (635), IsValidTime (638), IsValidDateTime (637), IsValidDateDay (635), IsValidDateMonthWeek (636)

Listing: ./examples/datutex/ex10.pp

```
Program Example10;

{ This program demonstrates the IsValidDateWeek function }

Uses SysUtils, DateUtils;

Var
  Y,W,D : Word;
  B : Boolean;

Begin
  For Y:=2000 to 2004 do
    begin
    B:=True;
    For W:=51 to 54 do
      For D:=1 to 7 do
        If B then
          begin
          B:=IsValidDateWeek(Y,W,D);
          If Not B then
            if (D=1) then
              Writeln(Y,' has exactly ',W,' weeks.')
            else
              Writeln(Y,' has ',W,' weeks and ',D-1,' days.');
          end;
    end;
End.
```

44.4.75 IsValidTime

Synopsis: Check whether a set of values is a valid time indication.

Declaration:
```
function IsValidTime(const AHour: Word; const AMinute: Word;
                     const ASecond: Word; const AMilliSecond: Word)
                      : Boolean
```

Visibility: default

Description: Check whether a set of values is a valid time indication.

44.4.76 JulianDateToDateTime

Synopsis: Convert a Julian date representation to a `TDateTime` value.

Declaration:
```
function JulianDateToDateTime(const AValue: Double) : TDateTime
```

Visibility: default

Description: `JulianDateToDateTime` converts the Julian `AValue` date/time indication to a regular `TDateTime` date/time indication.

See also: DateTimeToJulianDate (606), TryJulianDateToDateTime (678), DateTimeToModifiedJulianDate (606), TryModifiedJulianDateToDateTime (678)

44.4.77 LocalTimeToUniversal

Synopsis: Convert local time to UTC time.

Declaration:
```
function LocalTimeToUniversal(LT: TDateTime) : TDateTime
function LocalTimeToUniversal(LT: TDateTime; TZOffset: Integer)
                              : TDateTime
```

Visibility: default

Description: `LocalTimeToUniversal` converts a local time indication to a universal time indication: it undoes the `TZOffset` time zone offset from the `UT` Universal time (UTC). If no `TZOffset` is specified, the local time offset as returned by GetLocalTimeOffset (598) is used.

Note that for times in the past or in the future, or for time zones with DST, omitting the `TZoffset` may lead to wrong results depending on GetLocalTimeOffset being able to compute the correct offset for the `UT` on the target platform. Currently only Windows Vista and newer return correct offsets for a given date. Older Windows systems or Linux/Unix return always the offset for the current date.

See also: GetLocalTimeOffset (598), UniversalTimeToLocal (680)

44.4.78 MacTimeStampToUnix

Synopsis: Convert a Mac timestamp to a Unix timestamp.

Declaration:
```
function MacTimeStampToUnix(const AValue: Int64) : Int64
```

Visibility: default

Description: `MacTimeStampToUnix` converts the Mac timestamp indication in `AValue` to a UNIX timestamp indication (epoch time)

Errors: None.

See also: UnixTimeStampToMac (680), DateTimeToMac (606), MacToDateTime (639)

44.4.79 MacToDateTime

Synopsis: Convert a Mac timestamp to a `TDateTime` timestamp.

Declaration:
```
function MacToDateTime(const AValue: Int64) : TDateTime
```

Visibility: default

Description: `MacToDateTime` converts the Mac timestamp indication in `AValue` to a valid `TDateTime` indication.

Errors: None.

See also: UnixTimeStampToMac (680), DateTimeToMac (606), MacTimeStampToUnix (639)

44.4.80 MilliSecondOf

Synopsis: Extract the millisecond part from a TDateTime value.

Declaration: `function MilliSecondOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `MillisecondOf` returns the second of the minute part of the `AValue` date/time indication. It is a number between 0 and 999.

For an example, see YearOf (693)

See also: YearOf (693), WeekOf (680), MonthOf (648), DayOf (607), HourOf (622), MinuteOf (644), MilliSecondOf (640)

44.4.81 MilliSecondOfTheDay

Synopsis: Calculate the number of milliseconds elapsed since the start of the day.

Declaration: `function MilliSecondOfTheDay(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MilliSecondOfTheDay` returns the number of milliseconds that have passed since the start of the Day (00:00:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.000 will return 0.

For an example, see the HourOfTheDay (623) function.

See also: MilliSecondOfTheYear (642), MilliSecondOfTheMonth (641), MilliSecondOfTheWeek (642), MilliSecondOfTheHour (640), MilliSecondOfTheMinute (641), MilliSecondOfTheSecond (641), HourOfTheDay (623), MinuteOfTheDay (644), SecondOfTheDay (662)

44.4.82 MilliSecondOfTheHour

Synopsis: Calculate the number of milliseconds elapsed since the start of the hour.

Declaration: `function MilliSecondOfTheHour(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MilliSecondOfTheHour` returns the number of milliseconds that have passed since the start of the Hour (HH:00:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:00:00.000 will return 0.

For an example, see the MinuteOfTheHour (644) function.

See also: MilliSecondOfTheYear (642), MilliSecondOfTheMonth (641), MilliSecondOfTheWeek (642), MilliSecondOfTheDay (640), MilliSecondOfTheMinute (641), MilliSecondOfTheSecond (641), MinuteOfTheHour (644), SecondOfTheHour (662)

44.4.83 MilliSecondOfTheMinute

Synopsis: Calculate the number of milliseconds elapsed since the start of the minute.

Declaration: `function MilliSecondOfTheMinute(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MilliSecondOfTheMinute` returns the number of milliseconds that have passed since the start of the Minute (HH:MM:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:MM:00.000 will return 0.

For an example, see the SecondOfTheMinute (662) function.

See also: MilliSecondOfTheYear (642), MilliSecondOfTheMonth (641), MilliSecondOfTheWeek (642), MilliSecondOfTheDay (640), MilliSecondOfTheHour (640), MilliSecondOfTheMinute (641), MilliSecondOfTheSecond (641), SecondOfTheMinute (662)

44.4.84 MilliSecondOfTheMonth

Synopsis: Calculate number of milliseconds elapsed since the start of the month.

Declaration: `function MilliSecondOfTheMonth(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MilliSecondOfTheMonth` returns the number of milliseconds that have passed since the start of the month (00:00:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.000 on the first of the month will return 0.

For an example, see the WeekOfTheMonth (681) function.

See also: WeekOfTheMonth (681), DayOfTheMonth (608), HourOfTheMonth (623), MinuteOfTheMonth (645), SecondOfTheMonth (663), MilliSecondOfTheMonth (641)

44.4.85 MilliSecondOfTheSecond

Synopsis: Calculate the number of milliseconds elapsed since the start of the second.

Declaration: `function MilliSecondOfTheSecond(const AValue: TDateTime) : Word`

Visibility: default

Description: `MilliSecondOfTheSecond` returns the number of milliseconds that have passed since the start of the second (HH:MM:SS.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:MM:SS.000 will return 0.

See also: MilliSecondOfTheYear (642), MilliSecondOfTheMonth (641), MilliSecondOfTheWeek (642), MilliSecondOfTheDay (640), MilliSecondOfTheHour (640), MilliSecondOfTheMinute (641), SecondOfTheMinute (662)

Listing: ./examples/datutex/ex46.pp

```
Program Example46;

{ This program demonstrates the MilliSecondOfTheSecond function }

Uses SysUtils, DateUtils;

Var
```

```
  N : TDateTime;

Begin
  N:=Now;
  Writeln('MilliSecond of the Second : ',
          MilliSecondOfTheSecond(N));
End.
```

44.4.86 MilliSecondOfTheWeek

Synopsis: Calculate the number of milliseconds elapsed since the start of the week.

Declaration: `function MilliSecondOfTheWeek(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MilliSecondOfTheWeek` returns the number of milliseconds that have passed since the start of the Week (00:00:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.000 on the first of the Week will return 0.

For an example, see the DayOfTheWeek (608) function.

See also: MilliSecondOfTheYear (642), MilliSecondOfTheMonth (641), MilliSecondOfTheDay (640), MilliSecondOfTheHour (640), MilliSecondOfTheMinute (641), MilliSecondOfTheSecond (641), DayOfTheWeek (608), HourOfTheWeek (623), MinuteOfTheWeek (645), SecondOfTheWeek (663)

44.4.87 MilliSecondOfTheYear

Synopsis: Calculate the number of milliseconds elapsed since the start of the year.

Declaration: `function MilliSecondOfTheYear(const AValue: TDateTime) : Int64`

Visibility: default

Description: `MilliSecondOfTheYear` returns the number of milliseconds that have passed since the start of the year (January 1, 00:00:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. January 1 00:00:00.000 will return 0.

For an example, see the WeekOfTheYear (681) function.

See also: WeekOfTheYear (681), DayOfTheYear (608), HourOfTheYear (624), MinuteOfTheYear (646), SecondOfTheYear (663), MilliSecondOfTheYear (642)

44.4.88 MilliSecondsBetween

Synopsis: Calculate the number of whole milliseconds between two TDateTime values.

Declaration:
```
function MilliSecondsBetween(const ANow: TDateTime;
                             const AThen: TDateTime) : Int64
```

Visibility: default

Description: `MillisSecondsBetween` returns the number of whole milliseconds between `ANow` and `AThen`. This means a fractional part of a millisecond is dropped.

See also: YearsBetween (693), MonthsBetween (649), WeeksBetween (682), DaysBetween (609), HoursBetween (624), MinutesBetween (646), SecondsBetween (664)

Listing: ./examples/datutex/ex62.pp

```
Program Example62;

{ This program demonstrates the MilliSecondsBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of milliseconds between ');
  Write(TimeToStr(AThen),' and ',TimeToStr(ANow));
  Writeln(' : ',MilliSecondsBetween(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(0.9*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(1.0*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(1.1*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(2.5*OneMilliSecond);
  Test(D1,D2);
End.
```

44.4.89 MilliSecondSpan

Synopsis: Calculate the approximate number of milliseconds between two TDateTime values.

Declaration:
```
function MilliSecondSpan(const ANow: TDateTime; const AThen: TDateTime)
                         : Double
```

Visibility: default

Description: `MilliSecondSpan` returns the number of milliseconds between `ANow` and `AThen`. Since millisecond is the smallest fraction of a `TDateTime` indication, the returned number will always be an integer value.

See also: YearSpan (694), MonthSpan (649), WeekSpan (684), DaySpan (612), HourSpan (625), MinuteSpan (647), SecondSpan (664), MilliSecondsBetween (642)

Listing: ./examples/datutex/ex70.pp

```
Program Example70;

{ This program demonstrates the MilliSecondSpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
```

```
  Write('Number of milliseconds between ');
  Write(TimeToStr(AThen),' and ',TimeToStr(ANow));
  WriteIn(' : ',MilliSecondSpan(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(0.9*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(1.0*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(1.1*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(2.5*OneMilliSecond);
  Test(D1,D2);
End.
```

44.4.90 MinuteOf

Synopsis: Extract the minute part from a TDateTime value.

Declaration: `function MinuteOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `MinuteOf` returns the minute of the hour part of the `AValue` date/time indication. It is a number between 0 and 59.

For an example, see YearOf (693)

See also: YearOf (693), WeekOf (680), MonthOf (648), DayOf (607), HourOf (622), SecondOf (661), MilliSecondOf (640)

44.4.91 MinuteOfTheDay

Synopsis: Calculate the number of minutes elapsed since the start of the day.

Declaration: `function MinuteOfTheDay(const AValue: TDateTime) : Word`

Visibility: default

Description: `MinuteOfTheDay` returns the number of minutes that have passed since the start of the Day (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:59 will return 0.

For an example, see the HourOfTheDay (623) function.

See also: MinuteOfTheYear (646), MinuteOfTheMonth (645), MinuteOfTheWeek (645), MinuteOfTheHour (644), HourOfTheDay (623), SecondOfTheDay (662), MilliSecondOfTheDay (640)

44.4.92 MinuteOfTheHour

Synopsis: Calculate the number of minutes elapsed since the start of the hour.

Declaration: `function MinuteOfTheHour(const AValue: TDateTime) : Word`

Visibility: default

Description: `MinuteOfTheHour` returns the number of minutes that have passed since the start of the Hour (HH:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:00:59 will return 0.

See also: MinuteOfTheYear (646), MinuteOfTheMonth (645), MinuteOfTheWeek (645), MinuteOfTheDay (644), SecondOfTheHour (662), MilliSecondOfTheHour (640)

Listing: ./examples/datutex/ex44.pp

```
Program Example44;

{ This program demonstrates the MinuteOfTheHour function }

Uses SysUtils, DateUtils;

Var
  N : TDateTime;

Begin
  N:=Now;
  Writeln('Minute of the Hour       : ',MinuteOfTheHour(N));
  Writeln('Second of the Hour       : ',SecondOfTheHour(N));
  Writeln('MilliSecond of the Hour : ',
          MilliSecondOfTheHour(N));
End.
```

44.4.93 MinuteOfTheMonth

Synopsis: Calculate number of minutes elapsed since the start of the month.

Declaration: `function MinuteOfTheMonth(const AValue: TDateTime) : Word`

Visibility: default

Description: `MinuteOfTheMonth` returns the number of minutes that have passed since the start of the Month (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:59 on the first day of the month will return 0.

For an example, see the WeekOfTheMonth (681) function.

See also: WeekOfTheMonth (681), DayOfTheMonth (608), HourOfTheMonth (623), MinuteOfTheMonth (645), SecondOfTheMonth (663), MilliSecondOfTheMonth (641)

44.4.94 MinuteOfTheWeek

Synopsis: Calculate the number of minutes elapsed since the start of the week.

Declaration: `function MinuteOfTheWeek(const AValue: TDateTime) : Word`

Visibility: default

Description: `MinuteOfTheWeek` returns the number of minutes that have passed since the start of the week (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:59 on the first day of the week will return 0.

For an example, see the DayOfTheWeek (608) function.

See also: MinuteOfTheYear (646), MinuteOfTheMonth (645), MinuteOfTheDay (644), MinuteOfTheHour (644), DayOfTheWeek (608), HourOfTheWeek (623), SecondOfTheWeek (663), MilliSecondOfTheWeek (642)

44.4.95 MinuteOfTheYear

Synopsis: Calculate the number of minutes elapsed since the start of the year.

Declaration: `function MinuteOfTheYear(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MinuteOfTheYear` returns the number of minutes that have passed since the start of the year (January 1, 00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. January 1 00:00:59 will return 0.

For an example, see the WeekOfTheYear (681) function.

See also: WeekOfTheYear (681), DayOfTheYear (608), HourOfTheYear (624), MinuteOfTheYear (646), SecondOfTheYear (663), MilliSecondOfTheYear (642)

44.4.96 MinutesBetween

Synopsis: Calculate the number of whole minutes between two TDateTime values.

Declaration: `function MinutesBetween(const ANow: TDateTime; const AThen: TDateTime) : Int64`

Visibility: default

Description: `MinutesBetween` returns the number of whole minutes between `ANow` and `AThen`. This means the fractional part of a minute (seconds, milliseconds etc.) is dropped.

See also: YearsBetween (693), MonthsBetween (649), WeeksBetween (682), DaysBetween (609), HoursBetween (624), SecondsBetween (664), MilliSecondsBetween (642)

Listing: ./examples/datutex/ex60.pp

```
Program Example60;

{ This program demonstrates the MinutesBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of minutes between ');
  Write(TimeToStr(AThen),' and ',TimeToStr(ANow));
  Writeln(' : ',MinutesBetween(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(59*OneSecond);
```

```
  Test(D1,D2);
  D2:=D1-(61*OneSecond);
  Test(D1,D2);
  D2:=D1-(122*OneSecond);
  Test(D1,D2);
  D2:=D1-(306*OneSecond);
  Test(D1,D2);
  D2:=D1-(5.4*OneMinute);
  Test(D1,D2);
  D2:=D1-(2.5*OneMinute);
  Test(D1,D2);
End.
```

44.4.97 MinuteSpan

Synopsis: Calculate the approximate number of minutes between two TDateTime values.

Declaration:
```
function MinuteSpan(const ANow: TDateTime; const AThen: TDateTime)
                          : Double
```

Visibility: default

Description: `MinuteSpan` returns the number of minutes between `ANow` and `AThen`, including any fractional parts of a minute.

See also: YearSpan (694), MonthSpan (649), WeekSpan (684), DaySpan (612), HourSpan (625), SecondSpan (664), MilliSecondSpan (643), MinutesBetween (646)

Listing: ./examples/datutex/ex68.pp

```
Program Example68;

{ This program demonstrates the MinuteSpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
 Write('Number of minutes between ');
 Write(TimeToStr(AThen),' and ',TimeToStr(ANow));
 WriteIn(' : ',MinuteSpan(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(59*OneSecond);
  Test(D1,D2);
  D2:=D1-(61*OneSecond);
  Test(D1,D2);
  D2:=D1-(122*OneSecond);
  Test(D1,D2);
  D2:=D1-(306*OneSecond);
  Test(D1,D2);
```

```
  D2:=D1-(5.4*OneMinute);
  Test(D1,D2);
  D2:=D1-(2.5*OneMinute);
  Test(D1,D2);
End.
```

44.4.98 ModifiedJulianDateToDateTime

Synopsis: Convert a modified Julian date representation to a `TDateTime` value.

Declaration: `function ModifiedJulianDateToDateTime(const AValue: Double) : TDateTime`

Visibility: default

Description: Not yet implemented.

Errors: Currently, trying to use this function will raise an exception.

See also: DateTimeToJulianDate (606), JulianDateToDateTime (639), TryJulianDateToDateTime (678), DateTimeToModifiedJulianDate (606), TryModifiedJulianDateToDateTime (678)

44.4.99 MonthOf

Synopsis: Extract the month from a given date.

Declaration: `function MonthOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `MonthOf` returns the month part of the `AValue` date/time indication. It is a number between 1 and 12.

For an example, see YearOf (693)

See also: YearOf (693), DayOf (607), WeekOf (680), HourOf (622), MinuteOf (644), SecondOf (661), MilliSecondOf (640)

44.4.100 MonthOfTheYear

Synopsis: Extract the month of a TDateTime indication.

Declaration: `function MonthOfTheYear(const AValue: TDateTime) : Word`

Visibility: default

Description: `MonthOfTheYear` extracts the month part of `Avalue` and returns it. It is an alias for MonthOf (648), and is provided for completeness only, corresponding to the other `PartOfTheYear` functions.

For an example, see the WeekOfTheYear (681) function.

See also: MonthOf (648), WeekOfTheYear (681), DayOfTheYear (608), HourOfTheYear (624), MinuteOfTheYear (646), SecondOfTheYear (663), MilliSecondOfTheYear (642)

44.4.101 MonthsBetween

Synopsis: Calculate the number of whole months between two TDateTime values.

Declaration:
```
function MonthsBetween(const ANow: TDateTime; const AThen: TDateTime;
                       AExact: Boolean) : Integer
```

Visibility: default

Description: `MonthsBetween` returns the number of whole months between `ANow` and `AThen`. This number is an approximation, based on an average number of days of 30.4375 per month (average over 4 years). This means the fractional part of a month is dropped.

See also: YearsBetween (693), WeeksBetween (682), DaysBetween (609), HoursBetween (624), MinutesBetween (646), SecondsBetween (664), MilliSecondsBetween (642)

Listing: ./examples/datutex/ex56.pp

```
Program Example56;

{ This program demonstrates the MonthsBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of months between ');
  Write(DateToStr(AThen),' and ',DateToStr(ANow));
  Writeln(' : ',MonthsBetween(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Today;
  D2:=Today-364;
  Test(D1,D2);
  D2:=Today-365;
  Test(D1,D2);
  D2:=Today-366;
  Test(D1,D2);
  D2:=Today-390;
  Test(D1,D2);
  D2:=Today-368;
  Test(D1,D2);
  D2:=Today-1000;
  Test(D1,D2);
End.
```

44.4.102 MonthSpan

Synopsis: Calculate the approximate number of months between two TDateTime values.

Declaration:
```
function MonthSpan(const ANow: TDateTime; const AThen: TDateTime)
                   : Double
```

Visibility: default

Description: `MonthSpan` returns the number of month between `ANow` and `AThen`, including any fractional parts of a month. This number is an approximation, based on an average number of days of 30.4375 per month (average over 4 years).

See also: YearSpan (694), WeekSpan (684), DaySpan (612), HourSpan (625), MinuteSpan (647), SecondSpan (664), MilliSecondSpan (643), MonthsBetween (649)

Listing: ./examples/datutex/ex64.pp

```
Program Example64;

{ This program demonstrates the MonthSpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of months between ');
  Write(DateToStr(AThen),' and ',DateToStr(ANow));
  Writeln(' : ',MonthSpan(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Today;
  D2:=Today-364;
  Test(D1,D2);
  D2:=Today-365;
  Test(D1,D2);
  D2:=Today-366;
  Test(D1,D2);
  D2:=Today-390;
  Test(D1,D2);
  D2:=Today-368;
  Test(D1,D2);
  D2:=Today-1000;
  Test(D1,D2);
End.
```

44.4.103 NthDayOfWeek

Synopsis: Calculate which occurrence of weekday in the month a given day represents.

Declaration: `function NthDayOfWeek(const AValue: TDateTime) : Word`

Visibility: default

Description: `NthDayOfWeek` returns the occurrence of the weekday of `AValue` in the month. This is the N-th time that this weekday occurs in the month (e.g. the third Saturday of the month).

See also: EncodeDateMonthWeek (616), #rtl.sysutils.DayOfWeek (1699), DecodeDayOfWeekInMonth (615), EncodeDayOfWeekInMonth (617), TryEncodeDayOfWeekInMonth (674)

Listing: ./examples/datutex/ex104.pp

```
Program Example104;

{ This program demonstrates the NthDayOfWeek function }

Uses SysUtils, DateUtils;

Begin
  Write('Today is the ',NthDayOfWeek(Today),'-th ');
  WriteIn(formatdateTime('dddd',Today),' of the month.');
End.
```

44.4.104 PeriodBetween

Synopsis: Return the period (in years, months, days) between two dates.

Declaration:
```
procedure PeriodBetween(const ANow: TDateTime; const AThen: TDateTime;
                        out Years: Word; out months: Word;
                        out days: Word)
```

Visibility: default

Description: `PeriodBetween` returns the timespan between 2 dates (`ANow` and `AThen`), expressed as a number of years, months and days in the parameters `Years`, `months` and `days`. Only complete years, months and days are reported.

If `ANow` is before `AThen`, their values are reversed so the result is always positive.

See also: YearsBetween (693), MonthsBetween (649), WeeksBetween (682), DaysBetween (609)

44.4.105 PreviousDayOfWeek

Synopsis: Given a day of the week, return the previous day of the week.

Declaration:
```
function PreviousDayOfWeek(DayOfWeek: Word) : Word
```

Visibility: default

Description: `PreviousDayOfWeek` returns the previous day of the week. If the current day is the first day of the week (1) then the last day will be returned (7).

Remark Note that the days of the week are in ISO notation, i.e. 1-based.

See also: Yesterday (695)

Listing: ./examples/datutex/ex22.pp

```
Program Example22;

{ This program demonstrates the PreviousDayOfWeek function }

Uses SysUtils, DateUtils;

Var
  D : Word;

Begin
```

```
  For D:=1 to 7 do
    WriteLn('Previous day of ',D,' is : ',PreviousDayOfWeek(D));
End.
```

44.4.106 RecodeDate

Synopsis: Replace date part of a TDateTime value with another date.

Declaration:
```
function RecodeDate(const AValue: TDateTime; const AYear: Word;
                    const AMonth: Word; const ADay: Word) : TDateTime
```

Visibility: default

Description: RecodeDate replaces the date part of the timestamp AValue with the date specified in AYear, AMonth, ADay. All other parts (the time part) of the date/time stamp are left untouched.

Errors: If one of the AYear, AMonth, ADay values is not within a valid range then an EConvertError exception is raised.

See also: RecodeYear (658), RecodeMonth (656), RecodeDay (653), RecodeHour (654), RecodeMinute (655), RecodeSecond (656), RecodeDate (652), RecodeTime (657), RecodeDateTime (652)

Listing: ./examples/datutex/ex94.pp

```
Program Example94;

{ This program demonstrates the RecodeDate function }

Uses SysUtils, DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;

Begin
  S:=FormatDateTime(Fmt,RecodeDate(Now,2001,1,1));
  WriteLn('This moment on the first of the millenium : ',S);
End.
```

44.4.107 RecodeDateTime

Synopsis: Replace selected parts of a TDateTime value with other values.

Declaration:
```
function RecodeDateTime(const AValue: TDateTime; const AYear: Word;
                        const AMonth: Word; const ADay: Word;
                        const AHour: Word; const AMinute: Word;
                        const ASecond: Word; const AMilliSecond: Word)
                         : TDateTime
```

Visibility: default

Description: RecodeDateTime replaces selected parts of the timestamp AValue with the date/time values specified in AYear, AMonth, ADay, AHour, AMinute, ASecond and AMilliSecond. If any of these values equals the predefined constant RecodeLeaveFieldAsIs (600), then the corresponding part of the date/time stamp is left untouched.

Errors: If one of the values `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecondAMilliSecond` is not within a valid range (`RecodeLeaveFieldAsIs` excepted) then an `EConvertError` exception is raised.

See also: RecodeYear (658), RecodeMonth (656), RecodeDay (653), RecodeHour (654), RecodeMinute (655), RecodeSecond (656), RecodeMilliSecond (654), RecodeDate (652), RecodeTime (657), TryRecodeDateTime (679)

Listing: ./examples/datutex/ex96.pp

```
Program Example96;

{ This program demonstrates the RecodeDateTime function }

Uses SysUtils, DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;
  D : TDateTime ;

Begin
  D:=Now;
  D:=RecodeDateTime(D,2000,2,RecodeLeaveFieldAsIs,0,0,0,0);
  S:=FormatDateTime(Fmt,D);
  Writeln('This moment in februari 2000 : ',S);
End.
```

44.4.108 RecodeDay

Synopsis: Replace day part of a `TDateTime` value with another day.

Declaration:
```
function RecodeDay(const AValue: TDateTime; const ADay: Word)
                         : TDateTime
```

Visibility: default

Description: `RecodeDay` replaces the Day part of the timestamp `AValue` with `ADay`. All other parts of the date/time stamp are left untouched.

Errors: If the `ADay` value is not within a valid range (1 till the number of days in the month) then an `EConvertError` exception is raised.

See also: RecodeYear (658), RecodeMonth (656), RecodeHour (654), RecodeMinute (655), RecodeSecond (656), RecodeMilliSecond (654), RecodeDate (652), RecodeTime (657), RecodeDateTime (652)

Listing: ./examples/datutex/ex89.pp

```
Program Example89;

{ This program demonstrates the RecodeDay function }

Uses SysUtils, DateUtils;

Const
```

```
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;

Begin
  S:=FormatDateTime(Fmt,RecodeDay(Now,1));
  Writeln('This moment on the first of the month : ',S);
End.
```

44.4.109 RecodeHour

Synopsis: Replace hours part of a `TDateTime` value with another hour.

Declaration:
```
function RecodeHour(const AValue: TDateTime; const AHour: Word)
                          : TDateTime
```

Visibility: default

Description: `RecodeHour` replaces the Hour part of the timestamp `AValue` with `AHour`. All other parts of the date/time stamp are left untouched.

Errors: If the `AHour` value is not within a valid range (0..23) then an `EConvertError` exception is raised.

See also: RecodeYear (658), RecodeMonth (656), RecodeDay (653), RecodeMinute (655), RecodeSecond (656), RecodeMilliSecond (654), RecodeDate (652), RecodeTime (657), RecodeDateTime (652)

Listing: ./examples/datutex/ex90.pp

```
Program Example90;

{ This program demonstrates the RecodeHour function }

Uses SysUtils,DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;

Begin
  S:=FormatDateTime(Fmt,RecodeHour(Now,0));
  Writeln('Today, in the first hour : ',S);
End.
```

44.4.110 RecodeMilliSecond

Synopsis: Replace milliseconds part of a `TDateTime` value with another millisecond.

Declaration:
```
function RecodeMilliSecond(const AValue: TDateTime;
                           const AMilliSecond: Word) : TDateTime
```

Visibility: default

Description: `RecodeMilliSecond` replaces the millisecond part of the timestamp `AValue` with `AMilliSecond`. All other parts of the date/time stamp are left untouched.

Errors: If the `AMilliSecond` value is not within a valid range (0..999) then an `EConvertError` exception is raised.

See also: RecodeYear (658), RecodeMonth (656), RecodeDay (653), RecodeHour (654), RecodeMinute (655), RecodeSecond (656), RecodeDate (652), RecodeTime (657), RecodeDateTime (652)

Listing: ./examples/datutex/ex93.pp

```
Program Example93;

{ This program demonstrates the RecodeMilliSecond function }

Uses SysUtils, DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss.zzz';

Var
  S : AnsiString;

Begin
  S:=FormatDateTime(Fmt, RecodeMilliSecond(Now,0));
  Writeln('This moment, milliseconds stripped : ',S);
End.
```

44.4.111 RecodeMinute

Synopsis: Replace minutes part of a `TDateTime` value with another minute.

Declaration:
```
function RecodeMinute(const AValue: TDateTime; const AMinute: Word)
                      : TDateTime
```

Visibility: default

Description: `RecodeMinute` replaces the Minute part of the timestamp `AValue` with `AMinute`. All other parts of the date/time stamp are left untouched.

Errors: If the `AMinute` value is not within a valid range (0..59) then an `EConvertError` exception is raised.

See also: RecodeYear (658), RecodeMonth (656), RecodeDay (653), RecodeHour (654), RecodeSecond (656), RecodeMilliSecond (654), RecodeDate (652), RecodeTime (657), RecodeDateTime (652)

Listing: ./examples/datutex/ex91.pp

```
Program Example91;

{ This program demonstrates the RecodeMinute function }

Uses SysUtils, DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;
```

```
Begin
  S:=FormatDateTime(Fmt,RecodeMinute(Now,0));
  Writeln('This moment in the first minute of the hour: ',S);
End.
```

44.4.112 RecodeMonth

Synopsis: Replace month part of a `TDateTime` value with another month.

Declaration:
```
function RecodeMonth(const AValue: TDateTime; const AMonth: Word)
                          : TDateTime
```

Visibility: default

Description: `RecodeMonth` replaces the Month part of the timestamp `AValue` with `AMonth`. All other parts of the date/time stamp are left untouched.

Errors: If the `AMonth` value is not within a valid range (1..12) then an `EConvertError` exception is raised.

See also: RecodeYear (658), RecodeDay (653), RecodeHour (654), RecodeMinute (655), RecodeSecond (656), RecodeMilliSecond (654), RecodeDate (652), RecodeTime (657), RecodeDateTime (652)

Listing: ./examples/datutex/ex88.pp

```
Program Example88;

{ This program demonstrates the RecodeMonth function }

Uses SysUtils,DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;

Begin
  S:=FormatDateTime(Fmt,RecodeMonth(Now,5));
  Writeln('This moment in May : ',S);
End.
```

44.4.113 RecodeSecond

Synopsis: Replace seconds part of a `TDateTime` value with another second.

Declaration:
```
function RecodeSecond(const AValue: TDateTime; const ASecond: Word)
                          : TDateTime
```

Visibility: default

Description: `RecodeSecond` replaces the Second part of the timestamp `AValue` with `ASecond`. All other parts of the date/time stamp are left untouched.

Errors: If the `ASecond` value is not within a valid range (0..59) then an `EConvertError` exception is raised.

See also: RecodeYear (658), RecodeMonth (656), RecodeDay (653), RecodeHour (654), RecodeMinute (655), RecodeMilliSecond (654), RecodeDate (652), RecodeTime (657), RecodeDateTime (652)

Listing: ./examples/datutex/ex92.pp

```
Program Example92;

{ This program demonstrates the RecodeSecond function }

Uses SysUtils, DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;

Begin
  S:=FormatDateTime(Fmt,RecodeSecond(Now,0));
  Writeln('This moment, seconds stripped : ',S);
End.
```

44.4.114 RecodeTime

Synopsis: Replace time part of a `TDateTime` value with another time.

Declaration:
```
function RecodeTime(const AValue: TDateTime; const AHour: Word;
                    const AMinute: Word; const ASecond: Word;
                    const AMilliSecond: Word) : TDateTime
```

Visibility: default

Description: `RecodeTime` replaces the time part of the timestamp `AValue` with the date specified in `AHour`, `AMinute`, `ASecond` and `AMilliSecond`. All other parts (the date part) of the date/time stamp are left untouched.

Errors: If one of the values `AHour`, `AMinute`, `ASecondAMilliSecond` is not within a valid range then an `EConvertError` exception is raised.

See also: RecodeYear (658), RecodeMonth (656), RecodeDay (653), RecodeHour (654), RecodeMinute (655), RecodeSecond (656), RecodeMilliSecond (654), RecodeDate (652), RecodeDateTime (652)

Listing: ./examples/datutex/ex95.pp

```
Program Example95;

{ This program demonstrates the RecodeTime function }

Uses SysUtils, DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;

Begin
```

```
  S:=FormatDateTime(Fmt,RecodeTime(Now,8,0,0,0));
  Writeln('Today, 8 AM : ',S);
End.
```

44.4.115 RecodeYear

Synopsis: Replace year part of a `TDateTime` value with another year.

Declaration:
```
function RecodeYear(const AValue: TDateTime; const AYear: Word)
                          : TDateTime
```

Visibility: default

Description: `RecodeYear` replaces the year part of the timestamp `AValue` with `AYear`. All other parts of the date/time stamp are left untouched.

Errors: If the `AYear` value is not within a valid range (1..9999) then an `EConvertError` exception is raised.

See also: RecodeMonth (656), RecodeDay (653), RecodeHour (654), RecodeMinute (655), RecodeSecond (656), RecodeMilliSecond (654), RecodeDate (652), RecodeTime (657), RecodeDateTime (652)

Listing: ./examples/datutex/ex87.pp

```
Program Example87;

{ This program demonstrates the RecodeYear function }

Uses SysUtils,DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;

Begin
  S:=FormatDateTime(Fmt,RecodeYear(Now,1999));
  Writeln('This moment in 1999 : ',S);
End.
```

44.4.116 SameDate

Synopsis: Check whether two `TDateTime` values have the same date part.

Declaration:
```
function SameDate(const A: TDateTime; const B: TDateTime) : Boolean
```

Visibility: default

Description: `SameDate` compares the date parts of two timestamps `A` and `B` and returns `True` if they are equal, `False` if they are not.

The function simply checks whether CompareDate (601) returns zero.

See also: CompareDateTime (602), CompareDate (601), CompareTime (603), SameDateTime (659), SameTime (660)

Listing: ./examples/datutex/ex102.pp

```
Program Example102;

{ This program demonstrates the SameDate function }

Uses SysUtils, DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss.zzz';

Procedure Test(D1,D2 : TDateTime);

begin
  Write(FormatDateTime(Fmt,D1),' is the same date as ');
  Writeln(FormatDateTime(Fmt,D2),' : ',SameDate(D1,D2));
end;

Var
  D,N : TDateTime;

Begin
  D:=Today;
  N:=Now;
  Test(D,D);
  Test(N,N);
  Test(N+1,N);
  Test(N-1,N);
  Test(N+OneSecond,N);
  Test(N-OneSecond,N);
End.
```

44.4.117 SameDateTime

Synopsis: Check whether two `TDateTime` values have the same date and time parts.

Declaration: `function SameDateTime(const A: TDateTime; const B: TDateTime) : Boolean`

Visibility: default

Description: `SameDateTime` compares the date/time parts of two timestamps `A` and `B` and returns `True` if they are equal, `False` if they are not.

The function simply checks whether CompareDateTime (602) returns zero.

See also: CompareDateTime (602), CompareDate (601), CompareTime (603), SameDate (658), SameTime (660)

Listing: ./examples/datutex/ex101.pp

```
Program Example101;

{ This program demonstrates the SameDateTime function }

Uses SysUtils, DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss.zzz';
```

```
Procedure Test(D1,D2 : TDateTime);

begin
  Write(FormatDateTime(Fmt,D1),' is the same datetime as ');
  WriteIn(FormatDateTime(Fmt,D2),' : ',SameDateTime(D1,D2));
end;

Var
  D,N : TDateTime;

Begin
  D:=Today;
  N:=Now;
  Test(D,D);
  Test(N,N);
  Test(N+1,N);
  Test(N-1,N);
  Test(N+OneSecond,N);
  Test(N-OneSecond,N);
End.
```

44.4.118 SameTime

Synopsis: Check whether two `TDateTime` values have the same time part.

Declaration: `function SameTime(const A: TDateTime; const B: TDateTime) : Boolean`

Visibility: default

Description: `SameTime` compares the time parts of two timestamps `A` and `B` and returns `True` if they are equal, `False` if they are not.

The function simply checks whether CompareTime (603) returns zero.

See also: CompareDateTime (602), CompareDate (601), CompareTime (603), SameDateTime (659), SameDate (658)

Listing: ./examples/datutex/ex103.pp

```
Program Example102;

{ This program demonstrates the SameTime function }

Uses SysUtils,DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss.zzz';

Procedure Test(D1,D2 : TDateTime);

begin
  Write(FormatDateTime(Fmt,D1),' is the same time as ');
  WriteIn(FormatDateTime(Fmt,D2),' : ',SameTime(D1,D2));
end;

Var
  D,N : TDateTime;
```

```
Begin
  D:=Today;
  N:=Now;
  Test(D,D);
  Test(N,N);
  Test(N+1,N);
  Test(N-1,N);
  Test(N+OneSecond,N);
  Test(N-OneSecond,N);
End.
```

44.4.119 ScanDateTime

Synopsis: Scans a string for a TDateTime pattern and returns the date/time.

Declaration:
```
function ScanDateTime(const Pattern: string; const s: string;
                      const fmt: TFormatSettings; startpos: Integer)
                       : TDateTime;  Overload
function ScanDateTime(const Pattern: string; const s: string;
                      startpos: Integer) : TDateTime;  Overload
```

Visibility: default

Description: `ScanDateTime` scans string `S` for the date/time pattern `Pattern`, starting at position `StartPos` (default 1). Optionally, the format settings `fmt` can be specified.

In effect, this function does the opposite of what FormatDateTime (1743) does. The `Pattern` variable must contain a valid date/time pattern: note that not all possible formatdatetime patterns can be recognized, e.g., `hn` cannot be detected properly.

- There is a special placeholder: a '?' will match any character in the input string.
- Note that the / and : character will be replaced by the actual date and time separator.
- Similarly, the 'am/pm' and 'a/p' will be replaced with the actual AM/PM indicators.
- a TAB character (ASCII Code 9) will match any whitespace block

Errors: In case of an error, a EConvertError (1830) exception is raised.

See also: FormatDateTime (1743)

44.4.120 SecondOf

Synopsis: Extract the second part from a TDateTime value.

Declaration:
```
function SecondOf(const AValue: TDateTime) : Word
```

Visibility: default

Description: `SecondOf` returns the second of the minute part of the `AValue` date/time indication. It is a number between 0 and 59.

For an example, see YearOf (693)

See also: YearOf (693), WeekOf (680), MonthOf (648), DayOf (607), HourOf (622), MinuteOf (644), MilliSecondOf (640)

44.4.121 SecondOfTheDay

Synopsis: Calculate the number of seconds elapsed since the start of the day.

Declaration: `function SecondOfTheDay(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `SecondOfTheDay` returns the number of seconds that have passed since the start of the Day (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.999 return 0.

For an example, see the HourOfTheDay (623) function.

See also: SecondOfTheYear (663), SecondOfTheMonth (663), SecondOfTheWeek (663), SecondOfTheHour (662), SecondOfTheMinute (662), HourOfTheDay (623), MinuteOfTheDay (644), MilliSecondOfTheDay (640)

44.4.122 SecondOfTheHour

Synopsis: Calculate the number of seconds elapsed since the start of the hour.

Declaration: `function SecondOfTheHour(const AValue: TDateTime) : Word`

Visibility: default

Description: `SecondOfTheHour` returns the number of seconds that have passed since the start of the Hour (HH:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:00:00.999 return 0.

For an example, see the MinuteOfTheHour (644) function.

See also: SecondOfTheYear (663), SecondOfTheMonth (663), SecondOfTheWeek (663), SecondOfTheDay (662), SecondOfTheMinute (662), MinuteOfTheHour (644), MilliSecondOfTheHour (640)

44.4.123 SecondOfTheMinute

Synopsis: Calculate the number of seconds elapsed since the start of the minute.

Declaration: `function SecondOfTheMinute(const AValue: TDateTime) : Word`

Visibility: default

Description: `SecondOfTheMinute` returns the number of seconds that have passed since the start of the minute (HH:MM:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:MM:00.999 return 0.

See also: SecondOfTheYear (663), SecondOfTheMonth (663), SecondOfTheWeek (663), SecondOfTheDay (662), SecondOfTheHour (662), MilliSecondOfTheMinute (641)

Listing: ./examples/datutex/ex45.pp

```
Program Example45;

{ This program demonstrates the SecondOfTheMinute function }

Uses SysUtils, DateUtils;

Var
  N : TDateTime;
```

```
Begin
  N:=Now;
  Writeln('Second of the Minute       : ',SecondOfTheMinute(N));
  Writeln('MilliSecond of the Minute : ',
          MilliSecondOfTheMinute(N));
End.
```

44.4.124 SecondOfTheMonth

Synopsis: Calculate number of seconds elapsed since the start of the month.

Declaration: `function SecondOfTheMonth(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `SecondOfTheMonth` returns the number of seconds that have passed since the start of the month (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.999 on the first day of the month will return 0.

For an example, see the WeekOfTheMonth (681) function.

See also: WeekOfTheMonth (681), DayOfTheMonth (608), HourOfTheMonth (623), MinuteOfTheMonth (645), MilliSecondOfTheMonth (641)

44.4.125 SecondOfTheWeek

Synopsis: Calculate the number of seconds elapsed since the start of the week.

Declaration: `function SecondOfTheWeek(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `SecondOfTheWeek` returns the number of seconds that have passed since the start of the week (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.999 on the first day of the week will return 0.

For an example, see the DayOfTheWeek (608) function.

See also: SecondOfTheYear (663), SecondOfTheMonth (663), SecondOfTheDay (662), SecondOfTheHour (662), SecondOfTheMinute (662), DayOfTheWeek (608), HourOfTheWeek (623), MinuteOfTheWeek (645), MilliSecondOfTheWeek (642)

44.4.126 SecondOfTheYear

Synopsis: Calculate the number of seconds elapsed since the start of the year.

Declaration: `function SecondOfTheYear(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `SecondOfTheYear` returns the number of seconds that have passed since the start of the year (January 1, 00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. January 1 00:00:00.999 will return 0.

For an example, see the WeekOfTheYear (681) function.

See also: WeekOfTheYear (681), DayOfTheYear (608), HourOfTheYear (624), MinuteOfTheYear (646), SecondOfTheYear (663), MilliSecondOfTheYear (642)

44.4.127 SecondsBetween

Synopsis: Calculate the number of whole seconds between two TDateTime values.

Declaration:
```
function SecondsBetween(const ANow: TDateTime; const AThen: TDateTime)
                        : Int64
```

Visibility: default

Description: `SecondsBetween` returns the number of whole seconds between `ANow` and `AThen`. This means the fractional part of a second (milliseconds etc.) is dropped.

See also: YearsBetween (693), MonthsBetween (649), WeeksBetween (682), DaysBetween (609), HoursBetween (624), MinutesBetween (646), MilliSecondsBetween (642)

Listing: ./examples/datutex/ex61.pp

```
Program Example61;

{ This program demonstrates the SecondsBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of seconds between ');
  Write(TimeToStr(AThen),' and ',TimeToStr(ANow));
  WriteIn(' : ',SecondsBetween(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(999*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(1001*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(2001*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(5001*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(5.4*OneSecond);
  Test(D1,D2);
  D2:=D1-(2.5*OneSecond);
  Test(D1,D2);
End.
```

44.4.128 SecondSpan

Synopsis: Calculate the approximate number of seconds between two TDateTime values.

Declaration:
```
function SecondSpan(const ANow: TDateTime; const AThen: TDateTime)
                    : Double
```

Visibility: default

Description: `SecondSpan` returns the number of seconds between `ANow` and `AThen`, including any fractional parts of a second.

See also: YearSpan (694), MonthSpan (649), WeekSpan (684), DaySpan (612), HourSpan (625), MinuteSpan (647), MilliSecondSpan (643), SecondsBetween (664)

Listing: ./examples/datutex/ex69.pp

```
Program Example69;

{ This program demonstrates the SecondSpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of seconds between ');
  Write(TimeToStr(AThen),' and ',TimeToStr(ANow));
  Writeln(' : ',SecondSpan(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(999*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(1001*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(2001*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(5001*OneMilliSecond);
  Test(D1,D2);
  D2:=D1-(5.4*OneSecond);
  Test(D1,D2);
  D2:=D1-(2.5*OneSecond);
  Test(D1,D2);
End.
```

44.4.129 StartOfADay

Synopsis: Return the start of a day as a TDateTime value, given a day indication.

Declaration:
```
function StartOfADay(const AYear: Word; const AMonth: Word;
                     const ADay: Word) : TDateTime;  Overload
function StartOfADay(const AYear: Word; const ADayOfYear: Word)
                      : TDateTime;  Overload
```

Visibility: default

Description: `StartOfADay` returns a `TDateTime` value with the date/time indication of the start (0:0:0.000) of the day given by `AYear`, `AMonth`, `ADay`.

The day may also be indicated with a `AYear`, `ADayOfYear` pair.

See also: StartOfTheDay (668), StartOfTheWeek (669), StartOfAWeek (667), StartOfAMonth (666), StartOfTheMonth (668), EndOfTheWeek (621), EndOfAWeek (619), EndOfTheYear (622), EndOfAYear (620), EndOfTheMonth (621), EndOfAMonth (618), EndOfTheDay (620), EndOfADay (618)

Listing: ./examples/datutex/ex38.pp

```
Program Example38;

{ This program demonstrates the StartOfADay function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"Start of the day : "dd mmmm yyyy hh:nn:ss';

Var
  Y,M,D : Word;

Begin
  Y:=YearOf(Today);
  M:=MonthOf(Today);
  D:=DayOf(Today);
  WriteLn(FormatDateTime(Fmt,StartOfADay(Y,M,D)));
  DecodeDateDay(Today,Y,D);
  WriteLn(FormatDateTime(Fmt,StartOfADay(Y,D)));
End.
```

44.4.130 StartOfAMonth

Synopsis: Return first date of month, given a year/month pair.

Declaration:
```
function StartOfAMonth(const AYear: Word; const AMonth: Word)
                       : TDateTime
```

Visibility: default

Description: `StartOfAMonth` e returns a `TDateTime` value with the date of the first day of the month indicated by the `AYear`, `AMonth` pair.

See also: StartOfTheMonth (668), EndOfTheMonth (621), EndOfAMonth (618), EndOfTheYear (622), EndOfAYear (620), StartOfAWeek (667), StartOfTheWeek (669)

Listing: ./examples/datutex/ex30.pp

```
Program Example30;

{ This program demonstrates the StartOfAMonth function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"First day of this month : "dd mmmm yyyy';
Var
  Y,M : Word;

Begin
  Y:=YearOf(Today);
```

```
  M:=MonthOf(Today);
  WriteLn(FormatDateTime(Fmt,StartOfAMonth(Y,M)));
End.
```

44.4.131 StartOfAWeek

Synopsis: Return a day of the week, given a year, week and day in the week.

Declaration:
```
function StartOfAWeek(const AYear: Word; const AWeekOfYear: Word;
                      const ADayOfWeek: Word) : TDateTime
function StartOfAWeek(const AYear: Word; const AWeekOfYear: Word)
                      : TDateTime
```

Visibility: default

Description: `StartOfAWeek` returns a `TDateTime` value with the date of the indicated day of the week indicated by the `AYear`, `AWeek`, `ADayOfWeek` values.

The default value for `ADayOfWeek` is 1.

See also: StartOfTheWeek (669), EndOfTheWeek (621), EndOfAWeek (619), StartOfAMonth (666), EndOfTheYear (622), EndOfAYear (620), EndOfTheMonth (621), EndOfAMonth (618)

Listing: ./examples/datutex/ex34.pp

```
Program Example34;

{ This program demonstrates the StartOfAWeek function }

Uses SysUtils,DateUtils;

Const
  Fmt = '"First day of this week : "dd mmm yyyy hh:nn:ss';
  Fmt2 = '"Second day of this week : "dd mmm yyyy hh:nn:ss';

Var
  Y,W : Word;

Begin
  Y:=YearOf(Today);
  W:=WeekOf(Today);
  WriteLn(FormatDateTime(Fmt,StartOfAWeek(Y,W)));
  WriteLn(FormatDateTime(Fmt2,StartOfAWeek(Y,W,2)));
End.
```

44.4.132 StartOfAYear

Synopsis: Return the first day of a given year.

Declaration:
```
function StartOfAYear(const AYear: Word) : TDateTime
```

Visibility: default

Description: `StartOfAYear` returns a `TDateTime` value with the date of the first day of the year `AYear` (January 1).

See also: StartOfTheYear (669), EndOfTheYear (622), EndOfAYear (620), EndOfTheMonth (621), EndOfAMonth (618), StartOfAWeek (667), StartOfTheWeek (669)

Listing: ./examples/datutex/ex26.pp

```
Program Example26;

{ This program demonstrates the StartOfAYear function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"First day of this year : "dd mmmm yyyy';

Begin
  WriteLn(FormatDateTime(Fmt, StartOfAYear(YearOf(Today))));
End.
```

44.4.133 StartOfTheDay

Synopsis: Calculate the start of the day as a TDateTime value, given a moment in the day.

Declaration: `function StartOfTheDay(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `StartOfTheDay` extracts the date part of `AValue` and returns a `TDateTime` value with the date/time indication of the start (0:0:0.000) of this day.

See also: StartOfADay (665), StartOfTheWeek (669), StartOfAWeek (667), StartOfAMonth (666), StartOfTheMonth (668), EndOfTheWeek (621), EndOfAWeek (619), EndOfTheYear (622), EndOfAYear (620), EndOfTheMonth (621), EndOfAMonth (618), EndOftheDay (620), EndOfADay (618)

Listing: ./examples/datutex/ex36.pp

```
Program Example36;

{ This program demonstrates the StartOfTheDay function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"Start of the day : "dd mmmm yyyy hh:nn:ss';


Begin
  WriteLn(FormatDateTime(Fmt, StartOfTheDay(Today)));
End.
```

44.4.134 StartOfTheMonth

Synopsis: Calculate the first day of the month, given a date in that month.

Declaration: `function StartOfTheMonth(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `StartOfTheMonth` extracts the year and month parts of `AValue` and returns a `TDateTime` value with the date of the first day of that year and month as the StartOfAMonth (666) function.

See also: StartOfAMonth (666), EndOfTheYear (622), EndOfAYear (620), EndOfTheMonth (621), EndOfAMonth (618), StartOfAWeek (667), StartOfTheWeek (669)

Listing: ./examples/datutex/ex28.pp

```
Program Example28;

{ This program demonstrates the StartOfTheMonth function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"First day of this month : "dd mmmm yyyy';

Begin
  WriteLn(FormatDateTime(Fmt, StartOfTheMonth(Today)));
End.
```

44.4.135 StartOfTheWeek

Synopsis: Return the first day of the week, given a date.

Declaration: `function StartOfTheWeek(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `StartOfTheWeek` extracts the year and week parts of `AValue` and returns a `TDateTime` value with the date of the first day of that week as the StartOfAWeek (667) function.

See also: StartOfAWeek (667), EndOfTheWeek (621), EndOfAWeek (619), StartOfAMonth (666), EndOfTheYear (622), EndOfAYear (620), EndOfTheMonth (621), EndOfAMonth (618)

Listing: ./examples/datutex/ex32.pp

```
Program Example32;

{ This program demonstrates the StartOfTheWeek function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"First day of this week : "dd mmmm yyyy';

Begin
  WriteLn(FormatDateTime(Fmt, StartOfTheWeek(Today)));
End.
```

44.4.136 StartOfTheYear

Synopsis: Return the first day of the year, given a date in this year.

Declaration: `function StartOfTheYear(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `StartOfTheYear` extracts the year part of `AValue` and returns a `TDateTime` value with the date of the first day of that year (January 1), as the StartOfAYear (667) function.

See also: StartOfAYear (667), EndOfTheYear (622), EndOfAYear (620)

Listing: ./examples/datutex/ex24.pp

```
Program Example24;

{ This program demonstrates the StartOfTheYear function }

Uses SysUtils, DateUtils;

Const
  Fmt = '"First day of this year : "dd mmmm yyyy';

Begin
  WriteIn(FormatDateTime(Fmt, StartOfTheYear(Today)));
End.
```

44.4.137 TimeInRange

Synopsis: Checks whether a time value is in a given range.

Declaration:
```
function TimeInRange(ATime: TTime; AStartTime: TTime; AEndTime: TTime;
                     AInclusive: Boolean) : Boolean
```

Visibility: default

Description: `TimeInRange` checks whether the value `ATime` lies between `AStartTime` and `AEndTime`, and returns `True` if it is. When `AInclusive` is `True` (the default), then the limits are included. When `AInclusive` is `False`, the limits are excluded. Only the time part of the 3 parameters is considered.

Errors: The `AStartTime` value must be before `AEndTime`, but no check is performed.

See also: DateInRange (604), DateTimeInRange (605)

44.4.138 TimeOf

Synopsis: Extract the time part from a TDateTime indication.

Declaration:
```
function TimeOf(const AValue: TDateTime) : TDateTime
```

Visibility: default

Description: `TimeOf` extracts the time part from `AValue` and returns the result.

Since the `TDateTime` is actually a double with the time part encoded in the fractional part, this operation corresponds to a call to `Frac`.

See also: DateOf (604), YearOf (693), MonthOf (648), DayOf (607), HourOf (622), MinuteOf (644), SecondOf (661), MilliSecondOf (640)

Listing: ./examples/datutex/ex2.pp

```
Program Example2;

{ This program demonstrates the TimeOf function }

Uses SysUtils, DateUtils;

Begin
  WriteLn('Time is : ',TimeToStr(TimeOf(Now)));
End.
```

44.4.139 Today

Synopsis: Return the current date.

Declaration: `function Today : TDateTime`

Visibility: default

Description: `Today` is an alias for the Date (1694) function in the sysutils (1635) unit.

For an example, see Yesterday (695)

See also: Date (1694), Yesterday (695), Tomorrow (671)

44.4.140 Tomorrow

Synopsis: Return the next day.

Declaration: `function Tomorrow : TDateTime`

Visibility: default

Description: `Tomorrow` returns tomorrow's date. Tomorrow is determined from the system clock, i.e. it is Today (671) +1.

See also: Today (671), Yesterday (695)

Listing: ./examples/datutex/ex19.pp

```
Program Example19;

{ This program demonstrates the Tomorrow function }

Uses SysUtils, DateUtils;

Begin
  WriteLn(FormatDateTime('"Today is" dd mmmm yyyy',Today));
  WriteLn(FormatDateTime('"Tomorrow will be" dd mmmm yyyy',Tomorrow));
End.
```

44.4.141 TryEncodeDateDay

Synopsis: Encode a year and day of year to a `TDateTime` value.

Declaration:
```
function TryEncodeDateDay(const AYear: Word; const ADayOfYear: Word;
                          out AValue: TDateTime) : Boolean
```

Visibility: default

Description: `TryEncodeDateDay` encodes the values `AYear` and `ADayOfYear` to a date value and returns this value in `AValue`.

If the encoding was successful, `True` is returned. `False` is returned if any of the arguments is not valid.

See also: EncodeDateDay (616), EncodeDateTime (616), EncodeDateMonthWeek (616), EncodeDateWeek (617), TryEncodeDateTime (673), TryEncodeDateMonthWeek (672), TryEncodeDateWeek (674)

Listing: ./examples/datutex/ex84.pp

```
Program Example84;

{ This program demonstrates the TryEncodeDateDay function }

Uses SysUtils, DateUtils;

Var
  Y,DoY : Word;
  TS : TDateTime;

Begin
  DecodeDateDay(Now,Y,DoY);
  If TryEncodeDateDay(Y,DoY,TS) then
    Writeln('Today is : ',DateToStr(TS))
  else
    Writeln('Wrong year/day of year indication');
End.
```

44.4.142 TryEncodeDateMonthWeek

Synopsis: Encode a year, month, week of month and day of week to a `TDateTime` value.

Declaration:
```
function TryEncodeDateMonthWeek(const AYear: Word; const AMonth: Word;
                                 const AWeekOfMonth: Word;
                                 const ADayOfWeek: Word;
                                 out AValue: TDateTime) : Boolean
```

Visibility: default

Description: `TryEncodeDateTime` encodes the values `AYearAMonth`, `WeekOfMonth`,`ADayOfWeek`, to a date value and returns this value in `AValue`.

If the encoding was successful, `True` is returned, `False` if any of the arguments is not valid.

See also: DecodeDateMonthWeek (613), EncodeDateTime (616), EncodeDateWeek (617), EncodeDateDay (616), EncodeDateMonthWeek (616), TryEncodeDateTime (673), TryEncodeDateWeek (674), TryEncodeDateDay (671), NthDayOfWeek (650)

Listing: ./examples/datutex/ex86.pp

```
Program Example86;

{ This program demonstrates the TryEncodeDateMonthWeek function }

Uses SysUtils, DateUtils;
```

```
Var
  Y,M,Wom,Dow : Word;
  TS : TDateTime;

Begin
  DecodeDateMonthWeek(Now,Y,M,WoM,DoW);
  If TryEncodeDateMonthWeek(Y,M,WoM,Dow,TS) then
    WriteIn('Today is : ',DateToStr(TS))
  else
    WriteIn('Invalid year/month/week/dow indication');
End.
```

44.4.143 TryEncodeDateTime

Synopsis: Encode a Year, Month, Day, Hour, minute, seconds, milliseconds tuplet to a `TDateTime` value.

Declaration:
```
function TryEncodeDateTime(const AYear: Word; const AMonth: Word;
                           const ADay: Word; const AHour: Word;
                           const AMinute: Word; const ASecond: Word;
                           const AMilliSecond: Word;
                           out AValue: TDateTime) : Boolean
```

Visibility: default

Description: `EncodeDateTime` encodes the values `AYearAMonth`, `ADay`,`AHour`, `AMinute`,`ASecond` and `AMilliSecond` to a date/time value and returns this value in `AValue`.

If the date was encoded successfully, `True` is returned, `False` is returned if one of the arguments is not valid.

See also: EncodeDateTime (616), EncodeDateMonthWeek (616), EncodeDateWeek (617), EncodeDateDay (616), TryEncodeDateDay (671), TryEncodeDateWeek (674), TryEncodeDateMonthWeek (672)

Listing: ./examples/datutex/ex80.pp

```
Program Example79;

{ This program demonstrates the TryEncodeDateTime function }

Uses SysUtils, DateUtils;

Var
  Y,Mo,D,H,Mi,S,MS : Word;
  TS : TDateTime;

Begin
  DecodeDateTime(Now,Y,Mo,D,H,Mi,S,MS);
  If TryEncodeDateTime(Y,Mo,D,H,Mi,S,MS,TS) then
    WriteIn('Now is : ',DateTimeToStr(TS))
  else
    WriteIn('Wrong date/time indication');
End.
```

44.4.144 TryEncodeDateWeek

Synopsis: Encode a year, week and day of week triplet to a `TDateTime` value.

Declaration:
```
function TryEncodeDateWeek(const AYear: Word; const AWeekOfYear: Word;
                           out AValue: TDateTime; const ADayOfWeek: Word)
                            : Boolean
function TryEncodeDateWeek(const AYear: Word; const AWeekOfYear: Word;
                           out AValue: TDateTime) : Boolean
```

Visibility: default

Description: `TryEncodeDateWeek` encodes the values `AYear`, `AWeekOfYear` and `ADayOfWeek` to a date value and returns this value in `AValue`.

If the encoding was successful, `True` is returned. `False` is returned if any of the arguments is not valid.

See also: EncodeDateMonthWeek (616), EncodeDateWeek (617), EncodeDateTime (616), EncodeDateDay (616), TryEncodeDateTime (673), TryEncodeDateMonthWeek (672), TryEncodeDateDay (671)

Listing: ./examples/datutex/ex82.pp

```
Program Example82;

{ This program demonstrates the TryEncodeDateWeek function }

Uses SysUtils, DateUtils;

Var
  Y,W,Dow : Word;
  TS : TDateTime;

Begin
  DecodeDateWeek(Now,Y,W,Dow);
  If TryEncodeDateWeek(Y,W,TS,Dow) then
    WriteIn('Today is : ',DateToStr(TS))
  else
    WriteIn('Invalid date/week indication');
End.
```

44.4.145 TryEncodeDayOfWeekInMonth

Synopsis: Encode a year, month, week, day of week triplet to a `TDateTime` value.

Declaration:
```
function TryEncodeDayOfWeekInMonth(const AYear: Word;
                                   const AMonth: Word;
                                   const ANthDayOfWeek: Word;
                                   const ADayOfWeek: Word;
                                   out AValue: TDateTime) : Boolean
```

Visibility: default

Description: `EncodeDayOfWeekInMonth` encodes `AYear`, `AMonth`, `ADayOfweek` and `ANthDayOfweek` to a valid date stamp and returns the result in `AValue`.

`ANthDayOfweek`is the N-th time that this weekday occurs in the month, e.g. the third Saturday of the month.

The function returns `True` if the encoding was successful, `False` if any of the values is not in range.

See also: NthDayOfWeek (650), EncodeDateMonthWeek (616), #rtl.sysutils.DayOfWeek (1699), DecodeDayOfWeekInMonth (615), EncodeDayOfWeekInMonth (617)

Listing: ./examples/datutex/ex106.pp

```
Program Example105;

{ This program demonstrates the DecodeDayOfWeekInMonth function }

Uses SysUtils, DateUtils;

Var
  Y,M,NDoW,DoW : Word;
  D : TDateTime;
Begin
  DecodeDayOfWeekInMonth(Date,Y,M,NDoW,DoW);
  If TryEncodeDayOfWeekInMonth(Y,M,NDoW,DoW,D) then
    begin
    Write(DateToStr(D),' is the ',NDow,'-th ');
    WriteIn(formatdateTime('dddd',D),' of the month.');
    end
  else
    WriteIn('Invalid year/month/NthDayOfweek combination');
End.
```

44.4.146 TryEncodeTimeInterval

Synopsis: Try to encode an interval as a `TDateTime` value.

Declaration:
```
function TryEncodeTimeInterval(Hour: Word; Min: Word; Sec: Word;
                               MSec: Word; out Time: TDateTime) : Boolean
```

Visibility: default

Description: `TryEncodeTimeInterval` encodes a time interval expressed in `Hour`, `Min`, `Sec`, `MSec` as a `TDateTime` value and returns the value in `Time`. It returns `True` if `Min`, `Sec`, `MSec` contain valid time values (i.e. less than 60, 60 resp. MSec). The number of hours may be larger than 24.

See also: EncodeTimeInterval (617)

44.4.147 TryISO8601ToDate

Synopsis: Attempts to convert an ISO 8601-formatted date/time value to a TDateTime type.

Declaration:
```
function TryISO8601ToDate(const DateString: string;
                          out ADateTime: TDateTime; ReturnUTC: Boolean)
                           : Boolean
```

Visibility: default

Description: TryISO8601ToDate is a Boolean function which attempts to convert an ISO 8601-formatted date/time value to a TDateTime type.

`DateString` contains the ISO 8601 date/time value converted in the function. `DateString` must contain one of the supported ISO 8601 date/time notations supported in the routine. It is separated into date, time, and time zone values, and is converted by calling both the `TryISOStrToDateTime`

and TryISOTZStrToTZOffset functions. See TryISOStrToDateTime (676) for more information about supported ISO 8601 notations.

ReturnUTC indicates if the native date/time value needs to be adjusted to the UTC (Coordinated Universal Time) time zone. The default value for the argument is True, and causes GetLocalTimeOffset to be called to get and apply the time zone offset for the local computer.

ADateTime contains the native TDateTime value for the converted timestamp adjusted to the UTC time zone when ReturnUTC contains True.

Use DateToISO8601 to convert a native date/time value back to its representation in ISO 8601 notation.

See also: DateToISO8601 (607)

44.4.148 TryISOStrToDate

Synopsis: Attempts to convert an ISO 8601-formatted date value to a TDateTime type.

Declaration:
```
function TryISOStrToDate(const aString: string; out outDate: TDateTime)
                                 : Boolean
```

Visibility: default

Description: TryISOStrToDate is a Boolean function which attempts to convert an ISO 8601-formatted date value in aString to a TDateTime type. The return value is True if the string is successfully converted into a native date value.

aString contains the date value converted in the function, and can use one of the following ISO 8601 notations :

- YYYYMMDD
- YYYY-MM-DD

outDate is an output parameter where the converted TDateTime value is stored in the function. The return value is False (and outDate is set to an empty date value) if aString cannot be converted in the routine.The time portion of the TDateTime value is not used or updated in the function.

Use TryISOStrToTime to convert a time value to a native TDateTime type.

Use TryISOStrToDateTime to convert a string which contains both date and time values to a TDateTime type.

See also: TryISOStrToTime (677), TryISOStrToDateTime (676)

44.4.149 TryISOStrToDateTime

Synopsis: Attempts to convert an ISO 8601-formatted date/time value to a TDateTime type.

Declaration:
```
function TryISOStrToDateTime(const aString: string;
                                 out outDateTime: TDateTime) : Boolean
```

Visibility: default

Description: TryISOStrToDateTime is a Boolean function which attempts to convert an ISO 8601-formatted date/time value to a TDateTime type. aString contains the date/time value examined in the routine, and can use one of the supported ISO 8601 notations. Internally, TryISOStrToDateTime separates the value in aString into date and time parts and calls both TryISOStrToDate and TryISOStrToTime.

outDateTime is a TDateTime output parameter where the date/time value is stored in the function.

The return value is True if aString is successfully parsed and converted to a TDateTime type. The return value is False if aString contains a value that cannot be parsed in the function. When the return value is False, outDateTime is set to **0** (representing an empty date/time value).

Use TryISOStrToDate or TryISOStrToTime to convert a string using only a date or a time value (respectively).

Use TryISO8601ToDate to convert a string value which uses more intricate forms of the ISO 8601 time notation.

See also: TryISOStrToDate (676), TryISOStrToTime (677), TryISO8601ToDate (675)

44.4.150 TryISOStrToTime

Synopsis: Converts an ISO 8601-formatted time value to a TDateTime type.

Declaration:
```
function TryISOStrToTime(const aString: string; out outTime: TDateTime)
                                 : Boolean
```

Visibility: default

Description: TryISOStrToTime is a Boolean function which attempts to convert the specified ISO 8601 time value to a TDateTime type. aString contains the ISO 8601 time value examined in the function, and can use one of the following notations:

- HHNN
- HHNNSS
- HHNNSS.ZZZ
- HH:NN
- HH:NN:SS
- HH:NN:SS.ZZZ

aString may contain an optional time zone designation at the end of the string value, like **'Z'** for Zulu time zone or a time zone offset expressed using either positive or negative hours and minutes. For example:

- 16:00:00Z
- 12:00:00-04:00

The return value is True if the time value in aString is successfully converted in the function.

outTime is a TDateTime type used to store the time value converted in the function. When the return value is False, outTime contains **0** for an empty time value.

Use TryISOStrToDate to convert an ISO 8601 date value to a TDateTime type.

Use TryISOStrToDateTime to convert a string which contains both date and time values to a TDateTime type.

See also: TryISOStrToDate (676), TryISOStrToDateTime (676)

44.4.151 TryISOTZStrToTZOffset

Synopsis: Attempts to convert an ISO 8601 time zone designation to an offset in minutes.

Declaration:
```
function TryISOTZStrToTZOffset(const TZ: string; out TZOffset: Integer)
                                    : Boolean
```

Visibility: default

Description: `TryISOTZStrToTZOffset` is a `Boolean` function which attempts to convert the ISO 8601 time zone designation to a time zone offset expressed as a positive or negative number of minutes. `TZ` contains the time zone designation examined in the function. It should not contain any other portion of a date/time value using ISO 8601 notation - just the time zone designation.

`TryISOTZStrToTZOffset` recognizes the following values in the `TZ` argument:

ZRepresents time zone Zulu (short for "Zulu time") as used by the military and in navigation. Refers to UTC (Coordinated Universal Time), formerly known as Greenwich Mean Time.

±HHNN or ±HH:NN time zone designation expressed as a positive or negative number of hours and minutes for a given time value.

`TZOffset` is an Integer updated to contains the number of minutes needed to adjust a time value for a given time zone back to UTC (Zulu time). The hour and minute component values in TZ are converted to minutes and complemented (multiplied by -1) to derive the offset value. The derived value in TZOffset is 0 when TZ contains Zulu time, an empty string (''), or cannot be converted successfully.

The return value is `True` if the time zone designation in `TZ` is successfully converted to a time zone offset.

Use `ISOTZStrToTZOffset` to convert the time zone designation and raise an exception for an invalid time zone designation.

See also: ISOTZStrToTZOffset (598)

44.4.152 TryJulianDateToDateTime

Synopsis: Convert a Julian date representation to a `TDateTime` value.

Declaration:
```
function TryJulianDateToDateTime(const AValue: Double;
                                  out ADateTime: TDateTime) : Boolean
```

Visibility: default

Description: Try to convert a Julian date to a regular `TDateTime` date/time representation.

See also: DateTimeToJulianDate (606), JulianDateToDateTime (639), DateTimeToModifiedJulianDate (606), TryModifiedJulianDateToDateTime (678)

44.4.153 TryModifiedJulianDateToDateTime

Synopsis: Convert a modified Julian date representation to a `TDateTime` value.

Declaration:
```
function TryModifiedJulianDateToDateTime(const AValue: Double;
                                         out ADateTime: TDateTime)
                                          : Boolean
```

Visibility: default

Description: Not yet implemented.

Errors: Currently, trying to use this function will raise an exception.

See also: DateTimeToJulianDate (606), JulianDateToDateTime (639), TryJulianDateToDateTime (678), DateTimeToModifiedJulianDate (606), ModifiedJulianDateToDateTime (648)

44.4.154 TryRecodeDateTime

Synopsis: Replace selected parts of a `TDateTime` value with other values.

Declaration:
```
function TryRecodeDateTime(const AValue: TDateTime; const AYear: Word;
                           const AMonth: Word; const ADay: Word;
                           const AHour: Word; const AMinute: Word;
                           const ASecond: Word;
                           const AMilliSecond: Word;
                           out AResult: TDateTime) : Boolean
```

Visibility: default

Description: `TryRecodeDateTime` replaces selected parts of the timestamp `AValue` with the date/time values specified in `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecond` and `AMilliSecond`. If any of these values equals the predefined constant RecodeLeaveFieldAsIs (600), then the corresponding part of the date/time stamp is left untouched.

The resulting date/time is returned in `AValue`.

The function returns `True` if the encoding was successful. It returns `False` if one of the values `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecondAMilliSecond` is not within a valid range.

See also: RecodeYear (658), RecodeMonth (656), RecodeDay (653), RecodeHour (654), RecodeMinute (655), RecodeSecond (656), RecodeMilliSecond (654), RecodeDate (652), RecodeTime (657), RecodeDateTime (652)

Listing: ./examples/datutex/ex97.pp

```
Program Example97;

{ This program demonstrates the TryRecodeDateTime function }

Uses SysUtils, DateUtils;

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
  S : AnsiString;
  D : TDateTime ;

Begin
  If TryRecodeDateTime(Now,2000,2,RecodeLeaveFieldAsIs,0,0,0,0,D) then
    begin
    S:=FormatDateTime(Fmt,D);
    Writeln('This moment in februari 2000 : ',S);
    end
  else
    Writeln('This moment did/does not exist in februari 2000');
End.
```

44.4.155 UniversalTimeToLocal

Synopsis: Convert UTC time to local time.

Declaration:
```
function UniversalTimeToLocal(UT: TDateTime) : TDateTime
function UniversalTimeToLocal(UT: TDateTime; TZOffset: Integer)
                              : TDateTime
```

Visibility: default

Description: `UniversalTimeToLocal` converts a universal time indication to a local time: it applies the `TZOffset` time zone offset to the `UT` Universal time (UTC). If no `TZOffset` is specified, the local time offset as returned by GetLocalTimeOffset (598) is used.

Note that for times in the past or in the future, or for time zones with DST, omitting the `TZoffset` may lead to wrong results depending on GetLocalTimeOffset being able to compute the correct offset for the `LT` on the target platform. Currently only Windows Vista and newer return correct offsets for a given date. Older Windows systems or Linux/Unix return always the offset for the current date.

See also: GetLocalTimeOffset (598), LocalTimeToUniversal (639)

44.4.156 UnixTimeStampToMac

Synopsis: Convert Unix Timestamp to a Mac Timestamp.

Declaration:
```
function UnixTimeStampToMac(const AValue: Int64) : Int64
```

Visibility: default

Description: `UnixTimeStampToMac` converts the UNIX epoch time in `AValue` to a valid Mac timestamp indication and returns the result.

Errors: None.

See also: DateTimeToMac (606), MacToDateTime (639), MacTimeStampToUnix (639)

44.4.157 UnixToDateTime

Synopsis: Convert Unix epoch time to a `TDateTime` value.

Declaration:
```
function UnixToDateTime(const AValue: Int64; aReturnUTC: Boolean)
                        : TDateTime
```

Visibility: default

Description: `UnixToDateTime` converts epoch time (seconds elapsed since 1/1/1970) to a `TDateTime` value.

See also: DateTimeToUnix (606)

44.4.158 WeekOf

Synopsis: Extract week (of the year) from a given date.

Declaration:
```
function WeekOf(const AValue: TDateTime) : Word
```

Visibility: default

Description: `WeekOf` returns the week-of-the-year part of the `AValue` date/time indication. It is a number between 1 and 53.

For an example, see YearOf (693)

See also: YearOf (693), DayOf (607), MonthOf (648), HourOf (622), MinuteOf (644), SecondOf (661), MilliSecondOf (640)

44.4.159 WeekOfTheMonth

Synopsis: Extract the week of the month (and optionally month and year) from a TDateTime value.

Declaration:
```
function WeekOfTheMonth(const AValue: TDateTime) : Word;  Overload
function WeekOfTheMonth(const AValue: TDateTime; out AYear: Word;
                        out AMonth: Word) : Word;  Overload
```

Visibility: default

Description: `WeekOfTheMonth` extracts the week of the month from `Avalue` and returns it, and optionally returns the year and month as well (in `AYear`, `AMonth` respectively).

Remark Note that weeks are numbered from 1 using the ISO 8601 standard, and the day of the week as well. This means that the year and month may not be the same as the year part of the date, since the week may start in the previous year as the first week of the year is the week with at least 4 days in it.

See also: WeekOfTheYear (681), DayOfTheMonth (608), HourOfTheMonth (623), MinuteOfTheMonth (645), SecondOfTheMonth (663), MilliSecondOfTheMonth (641)

Listing: ./examples/datutex/ex41.pp

```
Program Example41;

{ This program demonstrates the WeekOfTheMonth function }

Uses SysUtils, DateUtils;

Var
  N : TDateTime;

Begin
  N:=Now;
  Writeln('Week of the Month        : ',WeekOfTheMonth(N));
  Writeln('Day of the Month         : ',DayOfTheMonth(N));
  Writeln('Hour of the Month        : ',HourOfTheMonth(N));
  Writeln('Minute of the Month      : ',MinuteOfTheMonth(N));
  Writeln('Second of the Month      : ',SecondOfTheMonth(N));
  Writeln('MilliSecond of the Month : ',
          MilliSecondOfTheMonth(N));
End.
```

44.4.160 WeekOfTheYear

Synopsis: Extract the week of the year (and optionally year) of a TDateTime indication.

Declaration:
```
function WeekOfTheYear(const AValue: TDateTime) : Word;  Overload
function WeekOfTheYear(const AValue: TDateTime; out AYear: Word) : Word
                       ;  Overload
```

Visibility: default

Description: `WeekOfTheYear` extracts the week of the year from `Avalue` and returns it, and optionally returns the year as well. It returns the same value as WeekOf (680).

Remark Note that weeks are numbered from 1 using the ISO 8601 standard, and the day of the week as well. This means that the year may not be the same as the year part of the date, since the week may start in the previous year as the first week of the year is the week with at least 4 days in it.

See also: WeekOf (680), MonthOfTheYear (648), DayOfTheYear (608), HourOfTheYear (624), MinuteOfTheYear (646), SecondOfTheYear (663), MilliSecondOfTheYear (642)

Listing: ./examples/datutex/ex40.pp

```
Program Example40;

{ This program demonstrates the WeekOfTheYear function }

Uses SysUtils, DateUtils;

Var
  N : TDateTime;

Begin
  N:=Now;
  Writeln('Month of the year        : ',MonthOfTheYear(N));
  Writeln('Week of the year         : ',WeekOfTheYear(N));
  Writeln('Day of the year          : ',DayOfTheYear(N));
  Writeln('Hour of the year         : ',HourOfTheYear(N));
  Writeln('Minute of the year       : ',MinuteOfTheYear(N));
  Writeln('Second of the year       : ',SecondOfTheYear(N));
  Writeln('MilliSecond of the year : ',
          MilliSecondOfTheYear(N));
End.
```

44.4.161 WeeksBetween

Synopsis: Calculate the number of whole weeks between two TDateTime values.

Declaration:
```
function WeeksBetween(const ANow: TDateTime; const AThen: TDateTime)
                      : Integer
```

Visibility: default

Description: `WeeksBetween` returns the number of whole weeks between `ANow` and `AThen`. This means the fractional part of a Week is dropped.

See also: YearsBetween (693), MonthsBetween (649), DaysBetween (609), HoursBetween (624), MinutesBetween (646), SecondsBetween (664), MilliSecondsBetween (642)

Listing: ./examples/datutex/ex57.pp

```
Program Example57;

{ This program demonstrates the WeeksBetween function }

Uses SysUtils, DateUtils;
```

```
Procedure Test(ANow,AThen : TDateTime);

begin
 Write('Number of weeks between ');
 Write(DateToStr(AThen),' and ',DateToStr(ANow));
 Writeln(' : ',WeeksBetween(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Today;
  D2:=Today-7;
  Test(D1,D2);
  D2:=Today-8;
  Test(D1,D2);
  D2:=Today-14;
  Test(D1,D2);
  D2:=Today-35;
  Test(D1,D2);
  D2:=Today-36;
  Test(D1,D2);
  D2:=Today-17;
  Test(D1,D2);
End.
```

44.4.162 WeeksInAYear

Synopsis: Return the number of weeks in a given year.

Declaration: `function WeeksInAYear(const AYear: Word) : Word`

Visibility: default

Description: `WeeksInAYear` returns the number of weeks in the year `AYear`. The return value is either 52 or 53.

Remark The first week of the year is determined according to the ISO 8601 standard: It is the first week that has at least 4 days in it, i.e. it includes a Thursday.

See also: WeeksInYear (684), DaysInYear (611), DaysInAYear (610), DaysInMonth (611), DaysInAMonth (610)

Listing: ./examples/datutex/ex13.pp

```
Program Example13;

{ This program demonstrates the WeeksInAYear function }

Uses SysUtils,DateUtils;

Var
  Y : Word;

Begin
  For Y:=1992 to 2010 do
    Writeln(Y,' has ',WeeksInAYear(Y),' weeks.');
End.
```

44.4.163 WeeksInYear

Synopsis: return the number of weeks in the year, given a date.

Declaration: `function WeeksInYear(const AValue: TDateTime) : Word`

Visibility: default

Description: `WeeksInYear` returns the number of weeks in the year part of `AValue`. The return value is either 52 or 53.

Remark The first week of the year is determined according to the ISO 8601 standard: It is the first week that has at least 4 days in it, i.e. it includes a Thursday.

See also: WeeksInAYear (683), DaysInYear (611), DaysInAYear (610), DaysInMonth (611), DaysInAMonth (610)

Listing: ./examples/datutex/ex12.pp

```
Program Example12;

{ This program demonstrates the WeeksInYear function }

Uses SysUtils, DateUtils;

Var
  Y : Word;

Begin
  For Y:=1992 to 2010 do
    WriteIn(Y,' has ',WeeksInYear(EncodeDate(Y,2,1)),' weeks.');
End.
```

44.4.164 WeekSpan

Synopsis: Calculate the approximate number of weeks between two TDateTime values.

Declaration:
```
function WeekSpan(const ANow: TDateTime; const AThen: TDateTime)
                  : Double
```

Visibility: default

Description: `WeekSpan` returns the number of weeks between `ANow` and `AThen`, including any fractional parts of a week.

See also: YearSpan (694), MonthSpan (649), DaySpan (612), HourSpan (625), MinuteSpan (647), SecondSpan (664), MilliSecondSpan (643), WeeksBetween (682)

Listing: ./examples/datutex/ex65.pp

```
Program Example57;

{ This program demonstrates the WeekSpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
```

```
  Write('Number of weeks between ');
  Write(DateToStr(AThen),' and ',DateToStr(ANow));
  WriteIn(' : ',WeekSpan(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Today;
  D2:=Today-7;
  Test(D1,D2);
  D2:=Today-8;
  Test(D1,D2);
  D2:=Today-14;
  Test(D1,D2);
  D2:=Today-35;
  Test(D1,D2);
  D2:=Today-36;
  Test(D1,D2);
  D2:=Today-17;
  Test(D1,D2);
End.
```

44.4.165 WithinPastDays

Synopsis: Check whether two TDateTimes are only a number of days apart.

Declaration:
```
function WithinPastDays(const ANow: TDateTime; const AThen: TDateTime;
                        const ADays: Integer) : Boolean
```

Visibility: default

Description: `WithinPastDays` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `ADays` days apart, or `False` if they are further apart.

Remark Since this function uses the DaysBetween (609) function to calculate the difference in days, this means that fractional days do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half days apart, the result will also be `True`

See also: WithinPastYears (692), WithinPastMonths (689), WithinPastWeeks (691), WithinPastHours (686), WithinPastMinutes (688), WithinPastSeconds (690), WithinPastMilliSeconds (687)

Listing: ./examples/datutex/ex50.pp

```
Program Example50;

{ This program demonstrates the WithinPastDays function }

Uses SysUtils,DateUtils;

Procedure Test(ANow,AThen : TDateTime; ADays : Integer);

begin
  Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));
  Write(' are within ',ADays,' days: ');
  WriteIn(WithinPastDays(ANow,AThen,ADays));
end;
```

```
Var
  D1,D2  :  TDateTime;

Begin
  D1:=Now;
  D2:=Today-23/24;
  Test(D1,D2,1);
  D2:=Today-1;
  Test(D1,D2,1);
  D2:=Today-25/24;
  Test(D1,D2,1);
  D2:=Today-26/24;
  Test(D1,D2,5);
  D2:=Today-5.4;
  Test(D1,D2,5);
  D2:=Today-2.5;
  Test(D1,D2,1);
  Test(D1,D2,2);
  Test(D1,D2,3);
End.
```

44.4.166 WithinPastHours

Synopsis: Check whether two TDateTimes are only a number of hours apart.

Declaration:
```
function WithinPastHours(const ANow: TDateTime; const AThen: TDateTime;
                         const AHours: Int64) : Boolean
```

Visibility: default

Description: `WithinPastHours` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `AHours` hours apart, or `False` if they are further apart.

Remark Since this function uses the HoursBetween (624) function to calculate the difference in Hours, this means that fractional hours do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half hours apart, the result will also be `True`

See also: WithinPastYears (692), WithinPastMonths (689), WithinPastWeeks (691), WithinPastDays (685), WithinPastMinutes (688), WithinPastSeconds (690), WithinPastMilliSeconds (687)

Listing: ./examples/datutex/ex51.pp

```
Program Example51;

{ This program demonstrates the WithinPastHours function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime; AHours : Integer);

begin
 Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));
 Write(' are within ',AHours,' hours: ');
 WriteIn(WithinPastHours(ANow,AThen,AHours));
end;

Var
```

```
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(59*OneMinute);
  Test(D1,D2,1);
  D2:=D1-(61*OneMinute);
  Test(D1,D2,1);
  D2:=D1-(122*OneMinute);
  Test(D1,D2,1);
  D2:=D1-(306*OneMinute);
  Test(D1,D2,5);
  D2:=D1-(5.4*OneHour);
  Test(D1,D2,5);
  D2:=D1-(2.5*OneHour);
  Test(D1,D2,1);
  Test(D1,D2,2);
  Test(D1,D2,3);
End.
```

44.4.167 WithinPastMilliSeconds

Synopsis: Check whether two TDateTimes are only a number of milliseconds apart.

Declaration:
```
function WithinPastMilliSeconds(const ANow: TDateTime;
                                const AThen: TDateTime;
                                const AMilliSeconds: Int64) : Boolean
```

Visibility: default

Description: `WithinPastMilliSeconds` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `AMilliSeconds` milliseconds apart, or `False` if they are further apart.

Remark Since this function uses the MilliSecondsBetween (642) function to calculate the difference in milliseconds, this means that fractional milliseconds do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half milliseconds apart, the result will also be `True`

See also: WithinPastYears (692), WithinPastMonths (689), WithinPastWeeks (691), WithinPastDays (685), WithinPastHours (686), WithinPastMinutes (688), WithinPastSeconds (690)

Listing: ./examples/datutex/ex54.pp

```
Program Example54;

{ This program demonstrates the WithinPastMilliSeconds function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime; AMilliSeconds : Integer);

begin
 Write(TimeToStr(AThen),' and ',TimeToStr(ANow));
 Write(' are within ',AMilliSeconds,' milliseconds: ');
 WriteIn(WithinPastMilliSeconds(ANow,AThen,AMilliSeconds));
end;

Var
```

```
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(0.9*OneMilliSecond);
  Test(D1,D2,1);
  D2:=D1-(1.0*OneMilliSecond);
  Test(D1,D2,1);
  D2:=D1-(1.1*OneMilliSecond);
  Test(D1,D2,1);
  D2:=D1-(2.5*OneMilliSecond);
  Test(D1,D2,1);
  Test(D1,D2,2);
  Test(D1,D2,3);
End.
```

44.4.168 WithinPastMinutes

Synopsis: Check whether two TDateTimes are only a number of minutes apart.

Declaration:
```
function WithinPastMinutes(const ANow: TDateTime;
                           const AThen: TDateTime; const AMinutes: Int64)
                            : Boolean
```

Visibility: default

Description: `WithinPastMinutes` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `AMinutes` minutes apart, or `False` if they are further apart.

Remark Since this function uses the MinutesBetween (646) function to calculate the difference in Minutes, this means that fractional minutes do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half minutes apart, the result will also be `True`

See also: WithinPastYears (692), WithinPastMonths (689), WithinPastWeeks (691), WithinPastDays (685), WithinPastHours (686), WithinPastSeconds (690), WithinPastMilliSeconds (687)

Listing: ./examples/datutex/ex52.pp

```
Program Example52;

{ This program demonstrates the WithinPastMinutes function }

Uses SysUtils,DateUtils;

Procedure Test(ANow,AThen : TDateTime; AMinutes : Integer);

begin
 Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));
 Write(' are within ',AMinutes,' Minutes: ');
 WriteIn(WithinPastMinutes(ANow,AThen,AMinutes));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(59*OneSecond);
```

```
  Test(D1,D2,1);
  D2:=D1-(61*OneSecond);
  Test(D1,D2,1);
  D2:=D1-(122*OneSecond);
  Test(D1,D2,1);
  D2:=D1-(306*OneSecond);
  Test(D1,D2,5);
  D2:=D1-(5.4*OneMinute);
  Test(D1,D2,5);
  D2:=D1-(2.5*OneMinute);
  Test(D1,D2,1);
  Test(D1,D2,2);
  Test(D1,D2,3);
End.
```

44.4.169 WithinPastMonths

Synopsis: Check whether two TDateTimes are only a number of months apart.

Declaration:
```
function WithinPastMonths(const ANow: TDateTime;
                          const AThen: TDateTime; const AMonths: Integer)
                           : Boolean
```

Visibility: default

Description: `WithinPastMonths` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `AMonths` months apart, or `False` if they are further apart.

Remark Since this function uses the MonthsBetween (649) function to calculate the difference in Months, this means that fractional months do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half months apart, the result will also be `True`

See also: WithinPastYears (692), WithinPastWeeks (691), WithinPastDays (685), WithinPastHours (686), WithinPastMinutes (688), WithinPastSeconds (690), WithinPastMilliSeconds (687)

Listing: ./examples/datutex/ex48.pp

```
Program Example48;

{ This program demonstrates the WithinPastMonths function }

Uses SysUtils,DateUtils;

Procedure Test(ANow,AThen : TDateTime; AMonths : Integer);

begin
  Write(DateToStr(AThen),' and ',DateToStr(ANow));
  Write(' are within ',AMonths,' months: ');
  WriteIn(WithinPastMonths(ANow,AThen,AMonths));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Today;
  D2:=Today-364;
  Test(D1,D2,12);
```

```
  D2:=Today-365;
  Test(D1,D2,12);
  D2:=Today-366;
  Test(D1,D2,12);
  D2:=Today-390;
  Test(D1,D2,12);
  D2:=Today-368;
  Test(D1,D2,11);
  D2:=Today-1000;
  Test(D1,D2,31);
  Test(D1,D2,32);
  Test(D1,D2,33);
End.
```

44.4.170 WithinPastSeconds

Synopsis: Check whether two TDateTimes are only a number of seconds apart.

Declaration:
```
function WithinPastSeconds(const ANow: TDateTime;
                           const AThen: TDateTime; const ASeconds: Int64)
                            : Boolean
```

Visibility: default

Description: `WithinPastSeconds` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `ASeconds` seconds apart, or `False` if they are further apart.

Remark Since this function uses the SecondsBetween (664) function to calculate the difference in seconds, this means that fractional seconds do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half seconds apart, the result will also be `True`

See also: WithinPastYears (692), WithinPastMonths (689), WithinPastWeeks (691), WithinPastDays (685), WithinPastHours (686), WithinPastMinutes (688), WithinPastMilliSeconds (687)

Listing: ./examples/datutex/ex53.pp

```
Program Example53;

{ This program demonstrates the WithinPastSeconds function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime; ASeconds : Integer);

begin
  Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));
  Write(' are within ',ASeconds,' seconds: ');
  WriteIn(WithinPastSeconds(ANow,AThen,ASeconds));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(999*OneMilliSecond);
  Test(D1,D2,1);
  D2:=D1-(1001*OneMilliSecond);
```

```
  Test(D1,D2,1);
  D2:=D1-(2001*OneMilliSecond);
  Test(D1,D2,1);
  D2:=D1-(5001*OneMilliSecond);
  Test(D1,D2,5);
  D2:=D1-(5.4*OneSecond);
  Test(D1,D2,5);
  D2:=D1-(2.5*OneSecond);
  Test(D1,D2,1);
  Test(D1,D2,2);
  Test(D1,D2,3);
End.
```

44.4.171 WithinPastWeeks

Synopsis: Check whether two TDateTimes are only a number of weeks apart.

Declaration:
```
function WithinPastWeeks(const ANow: TDateTime; const AThen: TDateTime;
                         const AWeeks: Integer) : Boolean
```

Visibility: default

Description: `WithinPastWeeks` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `AWeeks` weeks apart, or `False` if they are further apart.

Remark Since this function uses the WeeksBetween (682) function to calculate the difference in Weeks, this means that fractional Weeks do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half weeks apart, the result will also be `True`

See also: WithinPastYears (692), WithinPastMonths (689), WithinPastDays (685), WithinPastHours (686), WithinPastMinutes (688), WithinPastSeconds (690), WithinPastMilliSeconds (687)

Listing: ./examples/datutex/ex49.pp

```
Program Example49;

{ This program demonstrates the WithinPastWeeks function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime; AWeeks : Integer);

begin
  Write(DateToStr(AThen),' and ',DateToStr(ANow));
  Write(' are within ',AWeeks,' weeks: ');
  WriteIn(WithinPastWeeks(ANow,AThen,AWeeks));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Today;
  D2:=Today-7;
  Test(D1,D2,1);
  D2:=Today-8;
  Test(D1,D2,1);
  D2:=Today-14;
```

```
  Test(D1,D2,1);
  D2:=Today-35;
  Test(D1,D2,5);
  D2:=Today-36;
  Test(D1,D2,5);
  D2:=Today-17;
  Test(D1,D2,1);
  Test(D1,D2,2);
  Test(D1,D2,3);
End.
```

44.4.172 WithinPastYears

Synopsis: Check whether two TDateTimes are only a number of years apart.

Declaration:
```
function WithinPastYears(const ANow: TDateTime; const AThen: TDateTime;
                         const AYears: Integer) : Boolean
```

Visibility: default

Description: `WithinPastYears` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `AYears` years apart, or `False` if they are further apart.

Remark Since this function uses the YearsBetween (693) function to calculate the difference in years, this means that fractional years do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half years apart, the result will also be `True`

See also: WithinPastMonths (689), WithinPastWeeks (691), WithinPastDays (685), WithinPastHours (686), WithinPastMinutes (688), WithinPastSeconds (690), WithinPastMilliSeconds (687)

Listing: ./examples/datutex/ex47.pp

```
Program Example47;

{ This program demonstrates the WithinPastYears function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime; AYears : Integer);

begin
 Write(DateToStr(AThen),' and ',DateToStr(ANow));
 Write(' are within ',AYears,' years: ');
 WriteIn(WithinPastYears(ANow,AThen,AYears));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Today;
  D2:=Today-364;
  Test(D1,D2,1);
  D2:=Today-365;
  Test(D1,D2,1);
  D2:=Today-366;
  Test(D1,D2,1);
  D2:=Today-390;
```

```
  Test(D1,D2,1);
  D2:=Today-368;
  Test(D1,D2,1);
  D2:=Today-1000;
  Test(D1,D2,1);
  Test(D1,D2,2);
  Test(D1,D2,3);
End.
```

44.4.173 YearOf

Synopsis: Extract the year from a given date.

Declaration: `function YearOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `YearOf` returns the year part of the `AValue` date/time indication. It is a number between 1 and 9999.

See also: MonthOf (648), DayOf (607), WeekOf (680), HourOf (622), MinuteOf (644), SecondOf (661), MilliSecondOf (640)

Listing: ./examples/datutex/ex23.pp

```
Program Example23;

{ This program demonstrates the YearOf function }

Uses SysUtils, DateUtils;

Var
  D : TDateTime;

Begin
  D:=Now;
  Writeln('Year        : ',YearOf(D));
  Writeln('Month       : ',MonthOf(D));
  Writeln('Day         : ',DayOf(D));
  Writeln('Week        : ',WeekOf(D));
  Writeln('Hour        : ',HourOf(D));
  Writeln('Minute      : ',MinuteOf(D));
  Writeln('Second      : ',SecondOf(D));
  Writeln('MilliSecond : ',MilliSecondOf(D));
End.
```

44.4.174 YearsBetween

Synopsis: Calculate the number of whole years between two TDateTime values.

Declaration:
```
function YearsBetween(const ANow: TDateTime; const AThen: TDateTime;
                      AExact: Boolean) : Integer
```

Visibility: default

Description: `YearsBetween` returns the number of whole years between `ANow` and `AThen`. This number is an approximation, based on an average number of days of 365.25 per year (average over 4 years). This means the fractional part of a year is dropped.

See also: MonthsBetween (649), WeeksBetween (682), DaysBetween (609), HoursBetween (624), MinutesBetween (646), SecondsBetween (664), MilliSecondsBetween (642), YearSpan (694)

Listing: ./examples/datutex/ex55.pp

```
Program Example55;

{ This program demonstrates the YearsBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of years between ');
  Write(DateToStr(AThen),' and ',DateToStr(ANow));
  Writeln(' : ',YearsBetween(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Today;
  D2:=Today-364;
  Test(D1,D2);
  D2:=Today-365;
  Test(D1,D2);
  D2:=Today-366;
  Test(D1,D2);
  D2:=Today-390;
  Test(D1,D2);
  D2:=Today-368;
  Test(D1,D2);
  D2:=Today-1000;
  Test(D1,D2);
End.
```

44.4.175 YearSpan

Synopsis: Calculate the approximate number of years between two TDateTime values.

Declaration:
```
function YearSpan(const ANow: TDateTime; const AThen: TDateTime)
                  : Double
```

Visibility: default

Description: `YearSpan` returns the number of years between `ANow` and `AThen`, including any fractional parts of a year. This number is an approximation, based on an average number of days of 365.25 per year (average over 4 years).

See also: MonthSpan (649), WeekSpan (684), DaySpan (612), HourSpan (625), MinuteSpan (647), SecondSpan (664), MilliSecondSpan (643), YearsBetween (693)

Listing: ./examples/datutex/ex63.pp

```
Program Example63;

{ This program demonstrates the YearSpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin
  Write('Number of years between ');
  Write(DateToStr(AThen),' and ',DateToStr(ANow));
  Writeln(' : ',YearSpan(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Today;
  D2:=Today-364;
  Test(D1,D2);
  D2:=Today-365;
  Test(D1,D2);
  D2:=Today-366;
  Test(D1,D2);
  D2:=Today-390;
  Test(D1,D2);
  D2:=Today-368;
  Test(D1,D2);
  D2:=Today-1000;
  Test(D1,D2);
End.
```

44.4.176 Yesterday

Synopsis: Return the previous day.

Declaration: `function Yesterday : TDateTime`

Visibility: default

Description: `Yesterday` returns yesterday's date. Yesterday is determined from the system clock, i.e. it is Today (671) -1.

See also: Today (671), Tomorrow (671)

Listing: ./examples/datutex/ex18.pp

```
Program Example18;

{ This program demonstrates the Yesterday function }

Uses SysUtils, DateUtils;

Begin
  Writeln(FormatDateTime('"Today is " dd mmmm yyyy',Today));
```

```
  WriteLn(FormatDateTime('"Yesterday was " dd mmmm yyyy',Yesterday));
End.
```

44.5 TDateTimeHelper

44.5.1 Method overview

Page	Method	Description
699	AddDays	
699	AddHours	
699	AddMilliseconds	
699	AddMinutes	
699	AddMonths	
699	AddSeconds	
699	AddYears	
699	CompareTo	
697	Create	
697	CreateLocal	
697	CreateTotalSeconds	
697	CreateUnixTime	
701	DaysBetween	
699	EndOfDay	
698	EndOfMonth	
698	EndOfWeek	
698	EndOfYear	
700	Equals	
701	HoursBetween	
700	InRange	
700	IsAM	
700	IsInLeapYear	
700	IsPM	
700	IsSameDay	
700	IsToday	
701	MilliSecondsBetween	
701	MinutesBetween	
701	MonthsBetween	
701	SecondsBetween	
698	StartOfDay	
698	StartOfMonth	
698	StartOfWeek	
698	StartOfYear	
698	ToString	
701	WeeksBetween	
702	WithinDays	
702	WithinHours	
702	WithinMilliseconds	
702	WithinMinutes	
701	WithinMonths	
702	WithinSeconds	
702	WithinWeeks	
701	WithinYears	
700	YearsBetween	

44.5.2 Property overview

Page	Properties	Access	Description
703	Date	r	
704	Day	r	
703	DayOfWeek	r	
703	DayOfYear	r	
704	Hour	r	
704	Millisecond	r	
704	Minute	r	
704	Month	r	
702	Now	r	
704	Second	r	
703	Time	r	
703	Today	r	
703	Tomorrow	r	
705	TotalSeconds	r	
705	UnixTime	r	
704	Year	r	
703	Yesterday	r	

44.5.3 TDateTimeHelper.Create

Declaration:
```
class function Create(const aYear: Word; const aMonth: Word;
                      const aDay: Word) : TDateTime;  Overload;  Static
class function Create(const aYear: Word; const aMonth: Word;
                      const aDay: Word; const aHour: Word;
                      const aMinute: Word; const aSecond: Word;
                      const aMillisecond: Word) : TDateTime;  Overload
                      ;  Static
class function Create(Date: string; aFormat: string;
                      aDateSeparator: Char; aTimeSeparator: Char)
                       : TDateTime;  Overload;  Static
```

Visibility: public

44.5.4 TDateTimeHelper.CreateLocal

Declaration:
```
class function CreateLocal(Date: string; local: string) : TDateTime
                           ;  Static
```

Visibility: public

44.5.5 TDateTimeHelper.CreateUnixTime

Declaration:
```
class function CreateUnixTime(const Value: Int64) : TDateTime;  Static
```

Visibility: public

44.5.6 TDateTimeHelper.CreateTotalSeconds

Declaration:
```
class function CreateTotalSeconds(const Value: Int64) : TDateTime
                                  ;  Static
```

Visibility: public

44.5.7 TDateTimeHelper.ToString

Declaration: `function ToString(const aFormatStr: string) : string`

Visibility: public

44.5.8 TDateTimeHelper.StartOfYear

Declaration: `function StartOfYear : TDateTime`

Visibility: public

44.5.9 TDateTimeHelper.EndOfYear

Declaration: `function EndOfYear : TDateTime`

Visibility: public

44.5.10 TDateTimeHelper.StartOfMonth

Declaration: `function StartOfMonth : TDateTime`

Visibility: public

44.5.11 TDateTimeHelper.EndOfMonth

Declaration: `function EndOfMonth : TDateTime`

Visibility: public

44.5.12 TDateTimeHelper.StartOfWeek

Declaration: `function StartOfWeek : TDateTime`

Visibility: public

44.5.13 TDateTimeHelper.EndOfWeek

Declaration: `function EndOfWeek : TDateTime`

Visibility: public

44.5.14 TDateTimeHelper.StartOfDay

Declaration: `function StartOfDay : TDateTime`

Visibility: public

44.5.15 TDateTimeHelper.EndOfDay

Declaration: `function EndOfDay : TDateTime`

Visibility: public

44.5.16 TDateTimeHelper.AddYears

Declaration: `function AddYears(const aNumberOfYears: Integer) : TDateTime`

Visibility: public

44.5.17 TDateTimeHelper.AddMonths

Declaration: `function AddMonths(const aNumberOfMonths: Integer) : TDateTime`

Visibility: public

44.5.18 TDateTimeHelper.AddDays

Declaration: `function AddDays(const aNumberOfDays: Integer) : TDateTime`

Visibility: public

44.5.19 TDateTimeHelper.AddHours

Declaration: `function AddHours(const aNumberOfHours: Int64) : TDateTime`

Visibility: public

44.5.20 TDateTimeHelper.AddMinutes

Declaration: `function AddMinutes(const aNumberOfMinutes: Int64) : TDateTime`

Visibility: public

44.5.21 TDateTimeHelper.AddSeconds

Declaration: `function AddSeconds(const aNumberOfSeconds: Int64) : TDateTime`

Visibility: public

44.5.22 TDateTimeHelper.AddMilliseconds

Declaration: `function AddMilliseconds(const aNumberOfMilliseconds: Int64) : TDateTime`

Visibility: public

44.5.23 TDateTimeHelper.CompareTo

Declaration: `function CompareTo(const aDateTime: TDateTime) : TValueRelationship`

Visibility: public

44.5.24 TDateTimeHelper.Equals

Declaration: `function Equals(const aDateTime: TDateTime) : Boolean`

Visibility: public

44.5.25 TDateTimeHelper.IsSameDay

Declaration: `function IsSameDay(const aDateTime: TDateTime) : Boolean`

Visibility: public

44.5.26 TDateTimeHelper.InRange

Declaration:
```
function InRange(const aStartDateTime: TDateTime;
                 const aEndDateTime: TDateTime;
                 const aInclusive: Boolean) : Boolean
```

Visibility: public

44.5.27 TDateTimeHelper.IsInLeapYear

Declaration: `function IsInLeapYear : Boolean`

Visibility: public

44.5.28 TDateTimeHelper.IsToday

Declaration: `function IsToday : Boolean`

Visibility: public

44.5.29 TDateTimeHelper.IsAM

Declaration: `function IsAM : Boolean`

Visibility: public

44.5.30 TDateTimeHelper.IsPM

Declaration: `function IsPM : Boolean`

Visibility: public

44.5.31 TDateTimeHelper.YearsBetween

Declaration: `function YearsBetween(const aDateTime: TDateTime) : Integer`

Visibility: public

44.5.32 TDateTimeHelper.MonthsBetween

Declaration: `function MonthsBetween(const aDateTime: TDateTime) : Integer`

Visibility: public

44.5.33 TDateTimeHelper.WeeksBetween

Declaration: `function WeeksBetween(const aDateTime: TDateTime) : Integer`

Visibility: public

44.5.34 TDateTimeHelper.DaysBetween

Declaration: `function DaysBetween(const aDateTime: TDateTime) : Integer`

Visibility: public

44.5.35 TDateTimeHelper.HoursBetween

Declaration: `function HoursBetween(const aDateTime: TDateTime) : Int64`

Visibility: public

44.5.36 TDateTimeHelper.MinutesBetween

Declaration: `function MinutesBetween(const aDateTime: TDateTime) : Int64`

Visibility: public

44.5.37 TDateTimeHelper.SecondsBetween

Declaration: `function SecondsBetween(const aDateTime: TDateTime) : Int64`

Visibility: public

44.5.38 TDateTimeHelper.MilliSecondsBetween

Declaration: `function MilliSecondsBetween(const aDateTime: TDateTime) : Int64`

Visibility: public

44.5.39 TDateTimeHelper.WithinYears

Declaration:
```
function WithinYears(const aDateTime: TDateTime; const aYears: Integer)
                         : Boolean
```

Visibility: public

44.5.40 TDateTimeHelper.WithinMonths

Declaration:
```
function WithinMonths(const aDateTime: TDateTime;
                          const aMonths: Integer) : Boolean
```

Visibility: public

44.5.41 TDateTimeHelper.WithinWeeks

Declaration: `function WithinWeeks(const aDateTime: TDateTime; const aWeeks: Integer) : Boolean`

Visibility: public

44.5.42 TDateTimeHelper.WithinDays

Declaration: `function WithinDays(const aDateTime: TDateTime; const aDays: Integer) : Boolean`

Visibility: public

44.5.43 TDateTimeHelper.WithinHours

Declaration: `function WithinHours(const aDateTime: TDateTime; const aHours: Int64) : Boolean`

Visibility: public

44.5.44 TDateTimeHelper.WithinMinutes

Declaration: `function WithinMinutes(const aDateTime: TDateTime; const aMinutes: Int64) : Boolean`

Visibility: public

44.5.45 TDateTimeHelper.WithinSeconds

Declaration: `function WithinSeconds(const aDateTime: TDateTime; const aSeconds: Int64) : Boolean`

Visibility: public

44.5.46 TDateTimeHelper.WithinMilliseconds

Declaration: `function WithinMilliseconds(const aDateTime: TDateTime; const AMilliseconds: Int64) : Boolean`

Visibility: public

44.5.47 TDateTimeHelper.Now

Declaration: `Property Now : TDateTime`

Visibility: public

Access: Read

44.5.48 TDateTimeHelper.Today

Declaration: `Property Today : TDateTime`

Visibility: public

Access: Read

44.5.49 TDateTimeHelper.Yesterday

Declaration: `Property Yesterday : TDateTime`

Visibility: public

Access: Read

44.5.50 TDateTimeHelper.Tomorrow

Declaration: `Property Tomorrow : TDateTime`

Visibility: public

Access: Read

44.5.51 TDateTimeHelper.Date

Declaration: `Property Date : TDateTime`

Visibility: public

Access: Read

44.5.52 TDateTimeHelper.Time

Declaration: `Property Time : TDateTime`

Visibility: public

Access: Read

44.5.53 TDateTimeHelper.DayOfWeek

Declaration: `Property DayOfWeek : Word`

Visibility: public

Access: Read

44.5.54 TDateTimeHelper.DayOfYear

Declaration: `Property DayOfYear : Word`

Visibility: public

Access: Read

44.5.55 TDateTimeHelper.Year

Declaration: `Property Year : Word`

Visibility: public

Access: Read

44.5.56 TDateTimeHelper.Month

Declaration: `Property Month : Word`

Visibility: public

Access: Read

44.5.57 TDateTimeHelper.Day

Declaration: `Property Day : Word`

Visibility: public

Access: Read

44.5.58 TDateTimeHelper.Hour

Declaration: `Property Hour : Word`

Visibility: public

Access: Read

44.5.59 TDateTimeHelper.Minute

Declaration: `Property Minute : Word`

Visibility: public

Access: Read

44.5.60 TDateTimeHelper.Second

Declaration: `Property Second : Word`

Visibility: public

Access: Read

44.5.61 TDateTimeHelper.Millisecond

Declaration: `Property Millisecond : Word`

Visibility: public

Access: Read

44.5.62 TDateTimeHelper.UnixTime

Declaration: `Property UnixTime : Int64`

Visibility: public

Access: Read

44.5.63 TDateTimeHelper.TotalSeconds

Declaration: `Property TotalSeconds : Int64`

Visibility: public

Access: Read

Chapter 45

Reference for unit 'Dos'

45.1 Used units

Table 45.1: Used units by unit 'Dos'

Name	Page
BaseUnix	146
System	1362

45.2 Overview

The DOS unit gives access to some operating system calls related to files, the file system, date and time. Except for the PalmOS target, this unit is available to all supported platforms.

The unit was first written for Dos by Florian Klaempfl. It was ported to Linux by Mark May and enhanced by Michael Van Canneyt. The Amiga version was ported by Nils Sjoholm.

Under non-DOS systems, some of the functionality is lost, as it is either impossible or meaningless to implement it. Other than that, the functionality is the same for all operating systems.

Because the DOS unit is a Turbo Pascal compatibility unit, it is no longer actively developed: the interface is frozen and it is maintained only for the purpose of porting Turbo Pascal programs. For new development, it is recommended to use the sysutils (1635) unit instead.

45.3 System information.

Functions for retrieving and setting general system information such as date and time.

Table 45.2:

Name	Description
DosVersion (715)	Get OS version
GetCBreak (720)	Get setting of control-break handling flag
GetDate (721)	Get system date
GetIntVec (723)	Get interrupt vector status
GetTime (725)	Get system time
GetVerify (725)	Get verify flag
Intr (726)	Execute an interrupt
Keep (726)	Keep process in memory and exit
MSDos (726)	Execute MS-Dos function call
PackTime (727)	Pack time for file time
SetCBreak (727)	Set control-break handling flag
SetDate (728)	Set system date
SetIntVec (729)	Set interrupt vectors
SetTime (729)	Set system time
SetVerify (729)	Set verify flag
SwapVectors (730)	Swap interrupt vectors
UnPackTime (730)	Unpack file time

45.4 Process handling.

Functions to handle process information and starting new processes.

Table 45.3:

Name	Description
DosExitCode (714)	Exit code of last executed program
EnvCount (716)	Return number of environment variables
EnvStr (716)	Return environment string pair
Exec (717)	Execute program
GetEnv (721)	Return specified environment string

45.5 Directory and disk handling.

Routines to handle disk information.

Table 45.4:

Name	Description
AddDisk (712)	Add disk to list of disks (UNIX only)
DiskFree (713)	Return size of free disk space
DiskSize (714)	Return total disk size

45.6 File handling.

Routines to handle files on disk.

Table 45.5:

Name	Description
FExpand (717)	Expand filename to full path
FindClose (717)	Close finfirst/findnext session
FindFirst (718)	Start find of file
FindNext (719)	Find next file
FSearch (719)	Search for file in a path
FSplit (720)	Split filename in parts
GetFAttr (722)	Return file attributes
GetFTime (723)	Return file time
GetLongName (724)	Convert short filename to long filename (DOS only)
GetShortName (724)	Convert long filename to short filename (DOS only)
SetFAttr (728)	Set file attributes
SetFTime (729)	Set file time

45.7 File open mode constants.

These constants are used in the `Mode` field of the `TextRec` record. Gives information on the file-mode of the text I/O. For their definitions consult the following table:

Table 45.6: Possible mode constants

Constant	Description	Value
`fmclosed`	File is closed.	$D7B0
`fminput`	File is read only.	$D7B1
`fmoutput`	File is write only.	$D7B2
`fminout`	File is read and write.	$D7B3

45.8 File attributes.

The File Attribute constants are used in FindFirst (718), FindNext (719) to determine what type of special file to search for in addition to normal files. These flags are also used in the SetFAttr (728) and GetFAttr (722) routines to set and retrieve attributes of files. For their definitions consult fileattributes (708).

Table 45.7: Possible file attributes

Constant	Description	Value
`readonly`	Read-Only file attribute.	$01
`hidden`	Hidden file attribute.	$02
`sysfile`	System file attribute.	$04
`volumeid`	Volume ID file attribute.	$08
`directory`	Directory file attribute.	$10
`archive`	Archive file attribute.	$20
`anyfile`	Match any file attribute.	$3F

45.9 Constants, types and variables

45.9.1 Constants

`anyfile = $3F`

Match any file attribute.

`archive = $20`

Archive file attribute.

`directory = $10`

Directory file attribute.

`fauxiliary = $0010`

CPU auxiliary flag. Not used.

`fcarry = $0001`

CPU carry flag. Not used.

`FileNameLen = 255`

Maximum length of a filename.

`fmclosed = $D7B0`

File is closed.

`fminout = $D7B3`

File is read and write.

`fminput = $D7B1`

File is read only.

```
fmoutput = $D7B2
```

File is write only.

```
foverflow = $0800
```

CPU overflow flag. Not used.

```
fparity = $0004
```

CPU parity flag. Not used.

```
fsign = $0080
```

CPU sign flag. Not used.

```
fzero = $0040
```

CPU zero flag. Not used.

```
hidden = $02
```

Hidden file attribute.

```
readonly = $01
```

Read-Only file attribute.

```
sysfile = $04
```

System file attribute.

```
volumeid = $08
```

Volume ID file attribute.

45.9.2 Types

```
ComStr = string
```

Command-line string type.

```
DirStr = string
```

Full directory string type.

```
ExtStr = string
```

Filename extension string type.

```
NameStr = string
```

Fill filename string type.

```
PathStr = string
```

Full File path string type.

```
Registers = packed record
case i : Integer of
0: (
  ax : Word;
  f1 : Word;
  bx : Word;
  f2 : Word;
  cx : Word;
  f3 : Word;
  dx : Word;
  f4 : Word;
  bp : Word;
  f5 : Word;
  si : Word;
  f51 : Word;
  di : Word;
  f6 : Word;
  ds : Word;
  f7 : Word;
  es : Word;
  f8 : Word;
  flags : Word;
  fs : Word;
  gs : Word
  ;
);
1: (
  al : Byte;
  ah : Byte;
  f9 : Byte;
  f10 : Byte;
  bl
  : Byte;
  bh : Byte;
  f11 : Byte;
  f12 : Byte;
  cl : Byte;
  ch
  : Byte;
  f13 : Byte;
  f14 : Byte;
  dl : Byte;
  dh : Byte;
```

```
);
  2: (
  eax : LongInt;
  ebx : LongInt;
  ecx : LongInt;
  edx : LongInt
  ;
  ebp : LongInt;
  esi : LongInt;
  edi : LongInt;
);
end
```

This structure is only defined on a i386 compatible 32-bit platform, and is not used anywhere: it is defined for Turbo Pascal backwards compatibility only.

45.9.3 Variables

```
DosError : Integer
```

The `DosError` variable is used by the procedures in the Dos unit to report errors. It can have the following values :

Table 45.8: Dos error codes

Value	Meaning
2	File not found.
3	Path not found.
5	Access denied.
6	Invalid handle.
8	Not enough memory.
10	Invalid environment.
11	Invalid format.
18	No more files.

Other values are possible, but are not documented.

45.10 Procedures and functions

45.10.1 AddDisk

Synopsis: Add disk definition to list if drives (Unix only).

Declaration: `function AddDisk(const path: string) : Byte`

Visibility: default

Description: `AddDisk` adds a filename `S` to the internal list of disks. It is implemented for systems which do not use DOS type drive letters. This list is used to determine which disks to use in the DiskFree (713) and DiskSize (714) calls. The DiskFree (713) and DiskSize (714) functions need a file on the specified drive, since this is required for the `statfs` system call. The names are added sequentially. The Dos initialization code presets the first three disks to:

- `'.'` for the current drive,
- `'/fd0/.'` for the first floppy-drive (Linux only).
- `'/fd1/.'` for the second floppy-drive (Linux only).
- `'/'` for the first hard disk.

The first call to `AddDisk` will therefore add a name for the second harddisk, The second call for the third drive, and so on until 23 drives have been added (corresponding to drives `'D:'` to `'Z:'`)

Errors: None

See also: DiskFree (713), DiskSize (714)

45.10.2 DiskFree

Synopsis: Get free size on Disk.

Declaration: `function DiskFree(drive: Byte) : Int64`

Visibility: default

Description: `DiskFree` returns the number of free bytes on a disk. The parameter `Drive` indicates which disk should be checked. This parameter is 1 for floppy `a:`, 2 for floppy `b:`, etc. A value of 0 returns the free space on the current drive.

Remark For Unices: The `diskfree` and `disksize` functions need a file on the specified drive, since this is required for the `statfs` system call. These filenames are set in the initialization of the Dos unit, and have been preset to :

- `'.'` for the current drive,
- `'/fd0/.'` for the first floppy-drive (Linux only).
- `'/fd1/.'` for the second floppy-drive (Linux only).
- `'/'` for the first hard disk.

There is room for 1-26 drives. You can add a drive with the AddDisk (712) procedure. These settings can be coded in `dos.pp`, in the initialization part.

Errors: -1 when a failure occurs, or an invalid drive number is given.

See also: DiskSize (714), AddDisk (712)

Listing: ./examples/dosex/ex6.pp

```
Program Example6;
uses Dos;

{ Program to demonstrate the DiskSize and DiskFree function. }

begin
  WriteLn('This partition size has ',DiskSize(0),' bytes');
  WriteLn('Currently ',DiskFree(0),' bytes are free');
end.
```

45.10.3 DiskSize

Synopsis: Get total size of disk.

Declaration: `function DiskSize(drive: Byte) : Int64`

Visibility: default

Description: `DiskSize` returns the total size (in bytes) of a disk. The parameter `Drive` indicates which disk should be checked. This parameter is 1 for floppy `a:`, 2 for floppy `b:`, etc. A value of 0 returns the size of the current drive.

Remark For Unix only: The `diskfree` and `disksize` functions need a file on the specified drive, since this is required for the `statfs` system call. These filenames are set in the initialization of the Dos unit, and have been preset to :

- `'.'` for the current drive,
- `'/fd0/.'` for the first floppy-drive (Linux only).
- `'/fd1/.'` for the second floppy-drive (Linux only).
- `'/'` for the first hard disk.

There is room for 1-26 drives. You can add a drive with the AddDisk (712) procedure. These settings can be coded in `dos.pp`, in the initialization part.

For an example, see DiskFree (713).

Errors: -1 when a failure occurs, or an invalid drive number is given.

See also: DiskFree (713), AddDisk (712)

45.10.4 DosExitCode

Synopsis: Exit code of last executed program.

Declaration: `function DosExitCode : Word`

Visibility: default

Description: `DosExitCode` contains (in the low byte) the exit-code of a program executed with the `Exec` call.

Errors: None.

See also: Exec (717)

Listing: ./examples/dosex/ex5.pp

```
Program Example5;
uses Dos;

{ Program to demonstrate the Exec and DosExitCode function. }

begin
{$IFDEF Unix}
  WriteLn('Executing /bin/ls -la');
  Exec('/bin/ls','-la');
{$ELSE}
  WriteLn('Executing Dir');
  Exec(GetEnv('COMSPEC'),'/C dir');
{$ENDIF}
  WriteLn('Program returned with ExitCode ',Lo(DosExitCode));
end.
```

45.10.5 DosVersion

Synopsis: Current OS version.

Declaration: `function DosVersion : Word`

Visibility: default

Description: `DosVersion` returns the operating system or kernel version. The low byte contains the major version number, while the high byte contains the minor version number.

Remark On systems where versions consists of more then two numbers, only the first two numbers will be returned. For example Linux version 2.1.76 will give you DosVersion 2.1. Some operating systems, such as FreeBSD, do not have system calls to return the kernel version, in that case a value of 0 will be returned.

Errors: None.

Listing: ./examples/dosex/ex1.pp

```
Program Example1;
uses Dos;

{ Program to demonstrate the DosVersion function. }

var
  OS      : string[32];
  Version : word;
begin
{$IFDEF LINUX}
  OS:='Linux';
{$ENDIF}
{$ifdef FreeBSD}
  OS:='FreeBSD';
{$endif}
{$ifdef NetBSD}
  OS:='NetBSD';
{$endif}
{$ifdef Solaris}
  OS:='Solaris';
{$endif}
{$ifdef QNX}
  OS:='QNX';
{$endif}

{$IFDEF DOS}
  OS:='Dos';
{$ENDIF}
  Version:=DosVersion;
  WriteLn('Current ',OS,' version is ',Lo(Version),'.',Hi(Version));
end.
```

45.10.6 DTToUnixDate

Synopsis: Convert a DateTime to Unix timestamp.

Declaration: `function DTToUnixDate(DT: DateTime) : LongInt`

Visibility: default

Description: `DTToUnixDate` converts the DateTime value in `DT` to a Unix timestamp. It is an internal function, implemented on Unix platforms, and should not be used.

Errors: None.

See also: UnixDateToDT (730), PackTime (727), UnpackTime (730), GetTime (725), SetTime (729)

45.10.7 EnvCount

Synopsis: Return the number of environment variables.

Declaration: `function EnvCount : LongInt`

Visibility: default

Description: `EnvCount` returns the number of environment variables.

Errors: None.

See also: EnvStr (716), GetEnv (721)

45.10.8 EnvStr

Synopsis: Return environment variable by index.

Declaration: `function EnvStr(Index: LongInt) : string`

Visibility: default

Description: `EnvStr` returns the `Index`-th `Name=Value` pair from the list of environment variables. The index of the first pair is zero.

Errors: The length is limited to 255 characters.

See also: EnvCount (716), GetEnv (721)

Listing: ./examples/dosex/ex13.pp

```
Program Example13;
uses Dos;

{ Program to demonstrate the EnvCount and EnvStr function. }

var
  i : Longint;
begin
  WriteLn('Current Environment is:');
  for i:=1 to EnvCount do
   WriteLn(EnvStr(i));
end.
```

45.10.9 Exec

Synopsis: Execute another program, and wait for it to finish.

Declaration: `procedure Exec(const path: PathStr; const comline: ComStr)`

Visibility: default

Description: `Exec` executes the program in `Path`, with the options given by `ComLine`. The program name should *not* appear again in `ComLine`, it is specified in `Path`. Comline contains only the parameters that are passed to the program.

After the program has terminated, the procedure returns. The Exit value of the program can be consulted with the `DosExitCode` function.

For an example, see DosExitCode (714)

Errors: Errors are reported in `DosError`.

See also: DosExitCode (714)

45.10.10 FExpand

Synopsis: Expand a relative path to an absolute path.

Declaration: `function FExpand(const path: PathStr) : PathStr`

Visibility: default

Description: `FExpand` takes its argument and expands it to a complete filename, i.e. a filename starting from the root directory of the current drive, prepended with the drive-letter or volume name (when supported).

Remark On case sensitive file systems (such as Unix and Linux), the resulting name is left as it is, otherwise it is converted to uppercase.

Errors: FSplit (720)

Listing: ./examples/dosex/ex11.pp

```
Program Example11;
uses Dos;

{ Program to demonstrate the FExpand function. }

begin
  WriteLn('Expanded Name of this program is ',FExpand(ParamStr(0)));
end.
```

45.10.11 FindClose

Synopsis: Dispose resources allocated by a FindFirst (718)/FindNext (719) sequence.

Declaration: `procedure FindClose(var f: SearchRec)`

Visibility: default

Description: `FindClose` frees any resources associated with the search record `F`.

This call is needed to free any internal resources allocated by the FindFirst (718) or FindNext (719) calls.

The Unix implementation of the Dos unit therefore keeps a table of open directories, and when the table is full, closes one of the directories, and reopens another. This system is adequate but slow if you use a lot of `searchrecs`.

So, to speed up the findfirst/findnext system, the `FindClose` call was implemented. When you don't need a `searchrec` any more, you can tell this to the Dos unit by issuing a `FindClose` call. The directory which is kept open for this `searchrec` is then closed, and the table slot freed.

Remark It is recommended to use the Linux call `Glob` when looking for files on Linux.

Errors: Errors are reported in DosError.

See also: FindFirst (718), FindNext (719)

45.10.12 FindFirst

Synopsis: Start search for one or more files.

Declaration: `procedure FindFirst(const path: PathStr; attr: Word; var f: SearchRec)`

Visibility: default

Description: `FindFirst` searches the file specified in `Path`. Normal files, as well as all special files which have the attributes specified in `Attr` will be returned.

It returns a `SearchRec` record for further searching in `F`. `Path` can contain the wildcard characters `?` (matches any single character) and `*` (matches 0 ore more arbitrary characters). In this case `FindFirst` will return the first file which matches the specified criteria. If `DosError` is different from zero, no file(s) matching the criteria was(were) found.

Remark On the EMX target, you cannot issue two different `FindFirst` calls. That is, you must close any previous search operation with FindClose (717) before starting a new one. Failure to do so will end in a Run-Time Error 6 (Invalid file handle)

Errors: Errors are reported in DosError.

See also: FindNext (719), FindClose (717)

Listing: ./examples/dosex/ex7.pp

```
Program Example7;
uses Dos;

{ Program to demonstrate the FindFirst and FindNext function. }

var
  Dir : SearchRec;
begin
  FindFirst('*.*',archive,Dir);
  WriteLn('FileName'+Space(32),'FileSize':9);
  while (DosError=0) do
   begin
     WriteIn(Dir.Name+Space(40-Length(Dir.Name)),Dir.Size:9);
     FindNext(Dir);
   end;
  FindClose(Dir);
end.
```

45.10.13 FindNext

Synopsis: Find next matching file after FindFirst (718).

Declaration: `procedure FindNext(var f: SearchRec)`

Visibility: default

Description: `FindNext` takes as an argument a `SearchRec` from a previous `FindNext` call, or a `FindFirst` call, and tries to find another file which matches the criteria, specified in the `FindFirst` call. If `DosError` is different from zero, no more files matching the criteria were found.

For an example, see FindFirst (718).

Errors: `DosError` is used to report errors.

See also: FindFirst (718), FindClose (717)

45.10.14 FSearch

Synopsis: Search a file in searchpath.

Declaration: `function FSearch(path: PathStr; dirlist: string) : PathStr`

Visibility: default

Description: `FSearch` searches the file `Path` in all directories listed in `DirList`. The full name of the found file is returned. `DirList` must be a list of directories, separated by semi-colons. When no file is found, an empty string is returned.

Remark On Unix systems, `DirList` can also be separated by colons, as is customary on those environments.

Errors: None.

See also: FExpand (717)

Listing: ./examples/dosex/ex10.pp

```
program Example10;

uses Dos;

{ Program to demonstrate the FSearch function. }

var s:pathstr;

begin
  s:=FSearch(ParamStr(1),GetEnv('PATH'));
  if s='' then
    WriteLn(ParamStr(1),' not Found in PATH')
  else
    Writeln(ParamStr(1),' Found in PATH at ',s);
end.
```

45.10.15 FSplit

Synopsis: Split a full-path filename in parts.

Declaration:
```
procedure FSplit(path: PathStr; var dir: DirStr; var name: NameStr;
                 var ext: ExtStr)
```

Visibility: default

Description: `FSplit` splits a full file name into 3 parts : A `Path`, a `Name` and an extension (in `ext`.) The extension is taken to be all letters after the *last* dot (.). For Dos, however, an exception is made when `LFNSupport=False`, then the extension is defined as all characters after the *first* dot.

Errors: None.

See also: FSearch (719)

Listing: ./examples/dosex/ex12.pp

```
program Example12;

uses Dos;

{ Program to demonstrate the FSplit function. }

var dir:dirstr;
    name:namestr;
    ext:extstr;

begin
  FSplit(ParamStr(1),dir,name,ext);
  WriteLn('Splitted ',ParamStr(1),' in:');
  WriteLn('Path      : ',dir);
  WriteLn('Name      : ',name);
  WriteLn('Extension: ',ext);
end.
```

45.10.16 GetCBreak

Synopsis: Get control-Break flag.

Declaration:
```
procedure GetCBreak(var breakvalue: Boolean)
```

Visibility: default

Description: `GetCBreak` gets the status of CTRL-Break checking under Dos and Amiga. When `BreakValue` is `false`, then Dos only checks for the CTRL-Break key-press when I/O is performed. When it is set to `True`, then a check is done at every system call.

Remark Under non-Dos and non-Amiga operating systems, `BreakValue` always returns `True`.

Errors: None

See also: SetCBreak (727)

45.10.17 GetDate

Synopsis: Get the current date.

Declaration:
```
procedure GetDate(var year: Word; var month: Word; var mday: Word;
                  var wday: Word)
```

Visibility: default

Description: `GetDate` returns the system's date. `Year` is a number in the range 1980..2099.`mday` is the day of the month, `wday` is the day of the week, starting with Sunday as day 0.

Errors: None.

See also: GetTime (725), SetDate (728)

Listing: ./examples/dosex/ex2.pp

```
Program Example2;
uses Dos;

{ Program to demonstrate the GetDate function. }

const
  DayStr:array[0..6] of string[3]=('Sun','Mon','Tue','Wed','Thu','Fri','Sat');
  MonthStr:array[1..12] of string[3]=('Jan','Feb','Mar','Apr','May','Jun',
                                      'Jul','Aug','Sep','Oct','Nov','Dec');
var
  Year,Month,Day,WDay : word;
begin
  GetDate(Year,Month,Day,WDay);
  WriteLn('Current date');
  WriteLn(DayStr[WDay],', ',Day,' ',MonthStr[Month],' ',Year,'.');
end.
```

45.10.18 GetEnv

Synopsis: Get environment variable by name.

Declaration:
```
function GetEnv(envvar: string) : string
```

Visibility: default

Description: `Getenv` returns the value of the environment variable `EnvVar`. When there is no environment variable `EnvVar` defined, an empty string is returned.

Remark Under some operating systems (such as Unix), case is important when looking for `EnvVar`.

Errors: None.

See also: EnvCount (716), EnvStr (716)

Listing: ./examples/dosex/ex14.pp

```
Program Example14;
uses Dos;

{ Program to demonstrate the GetEnv function. }

begin
```

```
  WriteLn('Current PATH is ',GetEnv('PATH'));
end.
```

45.10.19 GetFAttr

Synopsis: Get file attributes.

Declaration: `procedure GetFAttr(var f; var attr: Word)`

Visibility: default

Description: `GetFAttr` returns the file attributes of the file-variable `f`. `F` can be a untyped or typed file, or of type `Text`. `f` must have been assigned, but not opened. The attributes can be examined with the following constants :

- `ReadOnly`
- `Hidden`
- `SysFile`
- `VolumeId`
- `Directory`
- `Archive`

Under Linux, supported attributes are:

- `Directory`
- `ReadOnly` if the current process doesn't have access to the file.
- `Hidden` for files whose name starts with a dot `('.')`.

Errors: Errors are reported in `DosError`

See also: SetFAttr (728)

Listing: ./examples/dosex/ex8.pp

```
Program Example8;
uses Dos;

{ Program to demonstrate the GetFAttr function. }

var
  Attr : Word;
  f    : File;
begin
  Assign(f,ParamStr(1));
  GetFAttr(f,Attr);
  WriteLn('File ',ParamStr(1),' has attribute ',Attr);
  if (Attr and archive)<>0 then WriteLn('- Archive');
  if (Attr and directory)<>0 then WriteLn('- Directory');
  if (Attr and readonly)<>0 then WriteLn('- Read-Only');
  if (Attr and sysfile)<>0 then WriteLn('- System');
  if (Attr and hidden)<>0 then WriteLn('- Hidden');
end.
```

45.10.20 GetFTime

Synopsis: Get file last modification time.

Declaration: `procedure GetFTime(var f; var time: LongInt)`

Visibility: default

Description: `GetFTime` returns the modification time of a file. This time is encoded and must be decoded with `UnPackTime`. `F` must be a file type, which has been assigned, and opened.

Errors: Errors are reported in `DosError`

See also: SetFTime (729), PackTime (727), UnPackTime (730)

Listing: ./examples/dosex/ex9.pp

```
Program Example9;
uses Dos;

{ Program to demonstrate the GetFTime function. }

Function L0(w:word):string;
var
  s : string;
begin
  Str(w,s);
  if w<10 then
   L0:='0'+s
  else
   L0:=s;
end;

var
  f    : File;
  Time : Longint;
  DT   : DateTime;
begin
  if Paramcount>0 then
    Assign(f,ParamStr(1))
  else
    Assign(f,'ex9.pp');
  Reset(f);
  GetFTime(f,Time);
  Close(f);
  UnPackTime(Time,DT);
  Write('File ',ParamStr(1),' is last modified on ');
  WriteLn(L0(DT.Month),'-',L0(DT.Day),'-',DT.Year,
          ' at ',L0(DT.Hour),':',L0(DT.Min));
end.
```

45.10.21 GetIntVec

Synopsis: Get interrupt vector.

Declaration: `procedure GetIntVec(intno: Byte; var vector: pointer)`

Visibility: default

Description: `GetIntVec` returns the address of interrupt vector `IntNo`.

Remark This call does nothing, it is present for compatibility only. Modern systems do not allow low level access to the hardware.

Errors: None.

See also: SetIntVec (729)

45.10.22 GetLongName

Synopsis: Get the long filename of a DOS 8.3 filename.

Declaration: `function GetLongName(var p: string) : Boolean`

Visibility: default

Description: This function is only implemented in the GO32V2 and Win32 versions of Free Pascal.

`GetLongName` changes the filename `p` to a long filename if the API call to do this is successful. The resulting string is the long file name corresponding to the short filename `p`.

The function returns `True` if the API call was successful, `False` otherwise.

This function should only be necessary when using the DOS extender under Windows 95 and higher.

Errors: If the API call was not successful, `False` is returned.

See also: GetShortName (724)

45.10.23 GetMsCount

Synopsis: Number of milliseconds since a starting point.

Declaration: `function GetMsCount : Int64`

Visibility: default

Description: `GetMSCount` returns a number of milliseconds elapsed since a certain moment in time. This moment in time is implementation dependent. This function is used for timing purposes: Subtracting the results of 2 subsequent calls to this function returns the number of milliseconds elapsed between the two calls.

This call is not very reliable, it is recommended to use some system specific calls for timings.

See also: GetTime (725)

45.10.24 GetShortName

Synopsis: Get the short (8.3) filename of a long filename.

Declaration: `function GetShortName(var p: string) : Boolean`

Visibility: default

Description: This function is only implemented in the GO32V2 and Win32 versions of Free Pascal.

`GetShortName` changes the filename `p` to a short filename if the API call to do this is successful. The resulting string is the short file name corresponding to the long filename `p`.

The function returns `True` if the API call was successful, `False` otherwise.

This function should only be necessary when using the DOS extender under Windows 95 and higher.

Errors: If the API call was not successful, `False` is returned.

See also: GetLongName (724)

45.10.25 GetTime

Synopsis: Return the current time.

Declaration:
```
procedure GetTime(var hour: Word; var minute: Word; var second: Word;
                  var sec100: Word)
```

Visibility: default

Description: `GetTime` returns the system's time. `Hour` is a on a 24-hour time scale. `sec100` is in hundredth of a second.

Remark Certain operating systems (such as Amiga), always set the `sec100` field to zero.

Errors: None.

See also: GetDate (721), SetTime (729)

Listing: ./examples/dosex/ex3.pp

```
Program Example3;
uses Dos;

{ Program to demonstrate the GetTime function. }

Function L0(w:word):string;
var
  s : string;
begin
  Str(w,s);
  if w<10 then
   L0:='0'+s
  else
   L0:=s;
end;

var
  Hour,Min,Sec,HSec : word;
begin
  GetTime(Hour,Min,Sec,HSec);
  WriteLn('Current time');
  WriteLn(L0(Hour),':',L0(Min),':',L0(Sec));
end.
```

45.10.26 GetVerify

Synopsis: Get verify flag.

Declaration:
```
procedure GetVerify(var verify: Boolean)
```

Visibility: default

Description: `GetVerify` returns the status of the verify flag under Dos. When `Verify` is `True`, then Dos checks data which are written to disk, by reading them after writing. If `Verify` is `False`, then data written to disk are not verified.

Remark Under non-Dos systems (excluding EMX applications running under vanilla DOS), Verify is always `True`.

Errors: None.

See also: SetVerify (729)

45.10.27 Intr

Synopsis: Execute interrupt.

Declaration: `procedure Intr(intno: Byte; var regs: Registers)`

Visibility: default

Description: `Intr` executes a software interrupt number `IntNo` (must be between 0 and 255), with processor registers set to `Regs`. After the interrupt call returned, the processor registers are saved in `Regs`.

Remark Under non-Dos operating systems, this call does nothing.

Errors: None.

See also: MSDos (726)

45.10.28 Keep

Synopsis: Terminate and stay resident.

Declaration: `procedure Keep(exitcode: Word)`

Visibility: default

Description: `Keep` terminates the program, but stays in memory. This is used for TSR (Terminate Stay Resident) programs which catch some interrupt. `ExitCode` is the same parameter as the `Halt` function takes.

Remark This call does nothing, it is present for compatibility only.

Errors: None.

45.10.29 MSDos

Synopsis: Execute MS-DOS system call.

Declaration: `procedure MSDos(var regs: Registers)`

Visibility: default

Description: `MSDos` executes an operating system call. This is the same as doing a `Intr` call with the interrupt number for an OS call.

Remark Under non-Dos operating systems, this call does nothing. On DOS systems, this calls interrupt $21.

Errors: None.

See also: Intr (726)

45.10.30 PackTime

Synopsis: Pack DateTime value to a packed-time format.

Declaration: `procedure PackTime(var t: DateTime; var p: LongInt)`

Visibility: default

Description: `UnPackTime` converts the date and time specified in `T` to a packed-time format which can be fed to `SetFTime`.

Errors: None.

See also: SetFTime (729), FindFirst (718), FindNext (719), UnPackTime (730)

Listing: ./examples/dosex/ex4.pp

```
Program Example4;
uses Dos;

{ Program to demonstrate the PackTime and UnPackTime functions. }

var
  DT   : DateTime;
  Time : longint;
begin
  with DT do
   begin
     Year:=2008;
     Month:=11;
     Day:=11;
     Hour:=11;
     Min:=11;
     Sec:=11;
   end;
  PackTime(DT,Time);
  WriteLn('Packed Time : ',Time);
  UnPackTime(Time,DT);
  WriteLn('Unpacked Again:');
  with DT do
   begin
     WriteLn('Year  ',Year);
     WriteLn('Month ',Month);
     WriteLn('Day   ',Day);
     WriteLn('Hour  ',Hour);
     WriteLn('Min   ',Min);
     WriteLn('Sec   ',Sec);
   end;
end.
```

45.10.31 SetCBreak

Synopsis: Set Control-Break flag status.

Declaration: `procedure SetCBreak(breakvalue: Boolean)`

Visibility: default

Description: `SetCBreak` sets the status of CTRL-Break checking. When `BreakValue` is `false`, then Dos only checks for the CTRL-Break key-press when I/O is performed. When it is set to `True`, then a check is done at every system call.

Remark Under non-Dos and non-Amiga operating systems, this call does nothing.

Errors: None.

See also: GetCBreak (720)

45.10.32 SetDate

Synopsis: Set system date.

Declaration: `procedure SetDate(year: Word; month: Word; day: Word)`

Visibility: default

Description: `SetDate` sets the system's internal date. `Year` is a number between 1980 and 2099.

Remark On a Unix machine, there must be root privileges, otherwise this routine will do nothing. On other Unix systems, this call currently does nothing.

Errors: None.

See also: GetDate (721), SetTime (729)

45.10.33 SetFAttr

Synopsis: Set file attributes.

Declaration: `procedure SetFAttr(var f; attr: Word)`

Visibility: default

Description: `SetFAttr` sets the file attributes of the file-variable `F`. `F` can be a untyped or typed file, or of type `Text`. `F` must have been assigned, but not opened. The attributes can be a sum of the following constants:

- `ReadOnly`
- `Hidden`
- `SysFile`
- `VolumeId`
- `Directory`
- `Archive`

Remark Under Unix like systems (such as Linux and BeOS) the call exists, but is not implemented, i.e. it does nothing.

Errors: Errors are reported in `DosError`.

See also: GetFAttr (722)

45.10.34 SetFTime

Synopsis: Set file modification time.

Declaration: `procedure SetFTime(var f; time: LongInt)`

Visibility: default

Description: `SetFTime` sets the modification time of a file, this time is encoded and must be encoded with `PackTime`. `F` must be a file type, which has been assigned, and opened.

Remark Under Unix like systems (such as Linux and BeOS) the call exists, but is not implemented, i.e. it does nothing.

Errors: Errors are reported in `DosError`

See also: GetFTime (723), PackTime (727), UnPackTime (730)

45.10.35 SetIntVec

Synopsis: Set interrupt vector.

Declaration: `procedure SetIntVec(intno: Byte; vector: pointer)`

Visibility: default

Description: `SetIntVec` sets interrupt vector `IntNo` to `Vector`. `Vector` should point to an interrupt procedure.

Remark This call does nothing, it is present for compatibility only.

Errors: None.

See also: GetIntVec (723)

45.10.36 SetTime

Synopsis: Set system time.

Declaration: `procedure SetTime(hour: Word; minute: Word; second: Word; sec100: Word)`

Visibility: default

Description: `SetTime` sets the system's internal clock. The `Hour` parameter is on a 24-hour time scale.

Remark On a Linux machine, there must be root privileges, otherwise this routine will do nothing. On other Unix systems, this call currently does nothing.

Errors: None.

See also: GetTime (725), SetDate (728)

45.10.37 SetVerify

Synopsis: Set verify flag.

Declaration: `procedure SetVerify(verify: Boolean)`

Visibility: default

Description: `SetVerify` sets the status of the verify flag under Dos. When `Verify` is `True`, then Dos checks data which are written to disk, by reading them after writing. If `Verify` is `False`, then data written to disk are not verified.

Remark Under non-Dos operating systems (excluding EMX applications running under vanilla Dos), Verify is always `True`.

Errors: None.

See also: SetVerify (729)

45.10.38 SwapVectors

Synopsis: Swap interrupt vectors.

Declaration: `procedure SwapVectors`

Visibility: default

Description: `SwapVectors` swaps the contents of the internal table of interrupt vectors with the current contents of the interrupt vectors. This is called typically in before and after an `Exec` call.

Remark Under certain operating systems, this routine may be implemented as an empty stub.

Errors: None.

See also: Exec (717), SetIntVec (729)

45.10.39 UnixDateToDt

Synopsis: Convert a Unix timestamp to a DateTime record.

Declaration: `procedure UnixDateToDt(SecsPast: LongInt; var Dt: DateTime)`

Visibility: default

Description: `DTToUnixDate` converts the Unix timestamp value in `SecsPast` to a DateTime representation in `DT`. It is an internal function, implemented on Unix platforms, and should not be used.

Errors: None.

See also: DTToUnixDate (715), PackTime (727), UnpackTime (730), GetTime (725), SetTime (729)

45.10.40 UnpackTime

Synopsis: Unpack packed file time to a DateTime value.

Declaration: `procedure UnpackTime(p: LongInt; var t: DateTime)`

Visibility: default

Description: `UnPackTime` converts the file-modification time in `p` to a `DateTime` record. The file-modification time can be returned by `GetFTime`, `FindFirst` or `FindNext` calls.

For an example, see PackTime (727).

Errors: None.

See also: GetFTime (723), FindFirst (718), FindNext (719), PackTime (727)

45.10.41 weekday

Synopsis: Return the day of the week.

Declaration: `function weekday(y: LongInt; m: LongInt; d: LongInt) : LongInt`

Visibility: default

Description: `WeekDay` returns the day of the week on which the day `Y/M/D` falls. Sunday is represented by 0, Saturday is 6.

Errors: On error, -1 is returned.

See also: PackTime (727), UnpackTime (730), GetTime (725), SetTime (729)

45.11 DateTime

```
DateTime = packed record
  Year : Word;
  Month : Word;
  Day : Word
  ;
  Hour : Word;
  Min : Word;
  Sec : Word;
end
```

The `DateTime` type is used in PackTime (727) and UnPackTime (730) for setting/reading file times with GetFTime (723) and SetFTime (729).

45.12 SearchRec

```
SearchRec = packed record
  SearchPos : TOff;
  SearchNum : LongInt
  ;
  DirPtr : Pointer;
  SearchType : Byte;
  SearchAttr : Byte;
  Mode : Word;
  Fill : Array[1..1] of Byte;
  Attr : Byte;
  Time
  : LongInt;
  Size : LongInt;
  Reserved : Word;
  Name : string;
  SearchSpec : string;
  NamePos : Word;
end
```

`SearchRec` is filled by the FindFirst (718) call and can be used in subsequent FindNext (719) calls to search for files. The structure of this record depends on the platform. Only the following fields are present on all platforms:

Attr File attributes.

Time File modification time.

Size File size

Name File name (name part only, no path)

Mode File access mode (Linux only)

Chapter 46

Reference for unit 'dxeload'

46.1 Used units

Table 46.1: Used units by unit 'dxeload'

Name	Page
System	1362

46.2 Overview

The dxeload unit was implemented by Pierre Mueller for dos, it allows to load a DXE file (an object file with 1 entry point) into memory and return a pointer to the entry point.

It exists only for dos.

46.3 Procedures and functions

46.3.1 dxe_load

Synopsis: Load DXE file in memory.

Declaration: `function dxe_load(filename: string) : pointer`

Visibility: default

Description: `dxe_load` loads the contents of the file `filename` into memory. It performs the necessary relocations in the object code, and returns then a pointer to the entry point of the code.

For an example, see the emu387 (738) unit in the RTL.

Errors: If an error occurs during the load or relocations, `Nil` is returned.

Chapter 47

Reference for unit 'dynlibs'

47.1 Used units

Table 47.1: Used units by unit 'dynlibs'

Name	Page
System	1362

47.2 Overview

The Dynlibs unit provides support for dynamically loading shared libraries. It is available only on those platforms that support shared libraries. The functionality available here may only be a part of the functionality available on each separate platform, in the interest of portability.

On UNIX platforms, using this unit will cause the program to be linked to the C library, as most shared libraries are implemented in C and the dynamical linker too.

47.3 Constants, types and variables

47.3.1 Constants

```
NilHandle = System.NilHandle
```

Correctly typed Nil handle - returned on error by LoadLibrary (736).

```
SharedSuffix = System.SharedSuffix
```

`SharedSuffix` contains the extension of a shared library (dynamically loadable library) on the current platform. It does not contain the . (dot) character. This can be used to determine the name of a shared library in a platform independent way.

47.3.2 Types

```
HModule = TLibHandle
```

Alias for TLibHandle (735) type.

```
TLibHandle = System.TLibHandle
```

`TLibHandle` should be considered an opaque type. It is defined differently on various platforms. The definition shown here depends on the platform for which the documentation was generated.

47.4 Procedures and functions

47.4.1 FreeLibrary

Synopsis: For compatibility with Delphi/Windows: Unload a library.

Declaration:
```
function FreeLibrary(Lib: TLibHandle) : Boolean
```

Visibility: default

Description: `FreeLibrary` provides the same functionality as UnloadLibrary (737), and is provided for compatibility with Delphi.

See also: UnloadLibrary (737)

47.4.2 GetLoadErrorStr

Synopsis: Return a description of the last error during load/get procedure address operations.

Declaration:
```
function GetLoadErrorStr : string
```

Visibility: default

Description: `GetLoadErrorStr` returns a textual description of the last library loading or unloading error, or a call to GetProcedureAddress (736). No other system calls may be made between the load call and the call of `GetLoadErrorStr`.

See also: LoadLibrary (736), GetProcedureAddress (736)

47.4.3 GetProcAddress

Synopsis: For compatibility with Delphi/Windows: Get the address of a procedure.

Declaration:
```
function GetProcAddress(Lib: TLibHandle; const ProcName: AnsiString)
                        : Pointer
```

Visibility: default

Description: `GetProcAddress` provides the same functionality as GetProcedureAddress (736), and is provided for compatibility with Delphi.

See also: GetProcedureAddress (736)

47.4.4 GetProcedureAddress

Synopsis: Get the address of a procedure or symbol in a dynamic library.

Declaration:
```
function GetProcedureAddress(Lib: TLibHandle;
                             const ProcName: AnsiString) : Pointer
function GetProcedureAddress(Lib: TLibHandle; Ordinal: TOrdinalEntry)
                             : Pointer
```

Visibility: default

Description: `GetProcedureAddress` returns a pointer to the location in memory of the symbol `ProcName` or ordinal value `Ordinal` in the dynamically loaded library specified by it's handle `lib`. If the symbol cannot be found or the handle is invalid, `Nil` is returned.

On Windows, only an exported procedure or function can be searched this way. On Unix platforms the location of any exported symbol can be retrieved this way.

Only windows and OS/2 support getting the address of a function using an ordinal value.

Errors: If the symbol cannot be found, `Nil` is returned.

See also: LoadLibrary (736), UnLoadLibrary (737)

47.4.5 LoadLibrary

Synopsis: Load a dynamic library and return a handle to it.

Declaration:
```
function LoadLibrary(const Name: RawByteString) : TLibHandle
function LoadLibrary(const Name: UnicodeString) : TLibHandle
```

Visibility: default

Description: `LoadLibrary` loads a dynamic library in file `Name` and returns a handle to it. If the library cannot be loaded, NilHandle (734) is returned.

No assumptions should be made about the location of the loaded library if a relative pathname is specified. The behaviour is dependent on the platform. Therefore it is best to specify an absolute pathname if possible.

Errors: On error, NilHandle (734) is returned.

See also: UnloadLibrary (737), GetProcedureAddress (736)

47.4.6 SafeLoadLibrary

Synopsis: Saves the control word and loads a library.

Declaration:
```
function SafeLoadLibrary(const Name: RawByteString) : TLibHandle
function SafeLoadLibrary(const Name: UnicodeString) : TLibHandle
```

Visibility: default

Description: `SafeLoadLibary` saves the FPU control word, and calls LoadLibrary (736) with library name `Name`. After that function has returned, the FPU control word is saved again. (only on Intel i386 CPUS).

See also: LoadLibrary (736)

47.4.7 UnloadLibrary

Synopsis: Unload a previously loaded library.

Declaration: `function UnloadLibrary(Lib: TLibHandle) : Boolean`

Visibility: default

Description: `UnloadLibrary` unloads a previously loaded library (specified by the handle `lib`). The call returns `True` if successful, `False` otherwise.

Errors: On error, `False` is returned.

See also: LoadLibrary (736), GetProcedureAddress (736)

Chapter 48

Reference for unit 'emu387'

48.1 Used units

Table 48.1: Used units by unit 'emu387'

Name	Page
System	1362

48.2 Overview

The emu387 unit was written by Pierre Mueller for dos. It sets up the coprocessor emulation for FPC under dos. It is not necessary to use this unit on other OS platforms because they either simply do not run on a machine without coprocessor, or they provide the coprocessor emulation themselves.

It shouldn't be necessary to use the function in this unit, it should be enough to place this unit in the `uses` clause of your program to enable the coprocessor emulation under dos. The unit initialization code will try and load the coprocessor emulation code and initialize it.

48.3 Procedures and functions

48.3.1 npxsetup

Synopsis: Set up coprocessor emulation.

Declaration: `procedure npxsetup(prog_name: string)`

Visibility: default

Description: `npxsetup` checks whether a coprocessor is found. If not, it loads the file wmemu387.dxe into memory and initializes the code in it.

If the environment variable `387` is set to `N`, then the emulation will be loaded, even if there is a coprocessor present. If the variable doesn't exist, or is set to any other value, the unit will try to detect the presence of a coprocessor unit.

The function searches the file wmemu387.dxe in the following way:

1. If the environment variable `EMU387` is set, then it is assumed to point at the wmemu387.dxe file.
2. if the environment variable `EMU387` does not exist, then the function will take the path part of `prog_name` and look in that directory for the file wmemu387.dxe.

It should never be necessary to call this function, because the initialization code of the unit contains a call to the function with as an argument `paramstr(0)`. This means that you should deliver the file `wmemu387.dxe` together with your program.

Errors: If there is an error, an error message is printed to standard error, and the program is halted, since any floating-point code is bound to fail anyhow.

Chapter 49

Reference for unit 'errors'

49.1 Used units

Table 49.1: Used units by unit 'errors'

Name	Page
System	1362
unixtype	2175

49.2 Overview

The errors unit contains routines to convert a UNIX system call error code to an error message: StrError (742). It is only available on UNIX platforms.

49.3 Constants, types and variables

49.3.1 Constants

```
sys_errlist : Array[0..sys_errn-1] of PChar = ('Success', 'Operation not permitted'
  , 'No such file or directory', 'No such process', 'Interrupted system call'
  , 'I/O error', 'No such device or address', 'Arg list too long', 'Exec format erro
  , 'Bad file number', 'No child processes', 'Try again', 'Out of memory'
  , 'Permission denied', 'Bad address', 'Block device required', 'Device or resource
  , 'File exists', 'Cross-device link', 'No such device', 'Not a directory'
  , 'Is a directory', 'Invalid argument', 'File table overflow', 'Too many open file
  , 'Not a typewriter', 'Text (code segment) file busy', 'File too large'
  , 'No space left on device', 'Illegal seek', 'Read-only file system'
  , 'Too many links', 'Broken pipe', 'Math argument out of domain of func'
  , 'Math result not representable', 'Resource deadlock would occur'
  , 'File name too long', 'No record locks available', 'Function not implemented'
  , 'Directory not empty', 'Too many symbolic links encountered', 'Operation would b
  , 'No message of desired type', 'Identifier removed', 'Channel number out of range
  , 'Level 2 not synchronized', 'Level 3 halted', 'Level 3 reset', 'Link number out
  , 'Protocol driver not attached', 'No CSI structure available', 'Level 2 halted'
```

```
  , 'Invalid exchange', 'Invalid request descriptor', 'Exchange full'
  , 'No anode', 'Invalid request code', 'Invalid slot', 'File locking deadlock error
  , 'Bad font file format', 'Device not a stream', 'No data available'
  , 'Timer expired', 'Out of streams resources', 'Machine is not on the network'
  , 'Package not installed', 'Object is remote', 'Link has been severed'
  , 'Advertise error', 'Srmount error', 'Communication error on send'
  , 'Protocol error', 'Multihop attempted', 'RFS specific error', 'Not a data messag
  , 'Value too large for defined data type', 'Name not unique on network'
  , 'File descriptor in bad state', 'Remote address changed', 'Can not access a need
  , 'Accessing a corrupted shared library', '.lib section in a.out corrupted'
  , 'Attempting to link in too many shared libraries', 'Cannot exec a shared library
  , 'Illegal byte sequence', 'Interrupted system call should be restarted'
  , 'Streams pipe error', 'Too many users', 'Socket operation on non-socket'
  , 'Destination address required', 'Message too long', 'Protocol wrong type for soc
  , 'Protocol not available', 'Protocol not supported', 'Socket type not supported'
  , 'Operation not supported on transport endpoint', 'Protocol family not supported'
  , 'Address family not supported by protocol', 'Address already in use'
  , 'Cannot assign requested address', 'Network is down', 'Network is unreachable'
  , 'Network dropped connection because of reset', 'Software caused connection abort
  , 'Connection reset by peer', 'No buffer space available', 'Transport endpoint is
  , 'Transport endpoint is not connected', 'Cannot send after transport endpoint shu
  , 'Too many references: cannot splice', 'Connection timed out', 'Connection refuse
  , 'Host is down', 'No route to host', 'Operation already in progress'
  , 'Operation now in progress', 'Stale NFS file handle', 'Structure needs cleaning'
  , 'Not a XENIX named type file', 'No XENIX semaphores available',
 'Is a named type file', 'Remote I/O error', 'Quota exceeded', 'No medium found'
  , 'Wrong medium type')
```

`sys_errn` is an array with then error codes for the current operating system. It should not be used directly, instead use StrError (742).

```
sys_errn = 125
```

`sys_errn` is the number of error codes for the current operating system. It should not be used directly, instead use StrError (742).

49.4 Procedures and functions

49.4.1 PError

Synopsis: Print error on standard error output.

Declaration: `procedure PError(const s: string; Errno: cint)`

Visibility: default

Description: `PError` will print the error message `S` followed by the `errNo` and the result for `StrError(errNo)` for `ErrNo` on standard output.

Errors: None.

See also: StrError (742)

49.4.2 StrError

Synopsis: Convert an error code to a string.

Declaration: `function StrError(err: cint) : string`

Visibility: default

Description: `StrError` will convert the error code `err` to a string.

Errors: If the error code is unknown or out of bounds, an 'Unknown error (err)' string will be returned.

See also: PError (741)

Chapter 50

Reference for unit 'exeinfo'

50.1 Used units

Table 50.1: Used units by unit 'exeinfo'

Name	Page
System	1362

50.2 Overview

The exeinfo unit implements some cross-platform routines to examine the contents of an executable: information about sections, mapping addresses to loaded modules etc.

It is mainly used by the lineinfo (973) and lnfodwrf (1006) unit to examine the binary for debug info.

50.3 Constants, types and variables

50.3.1 Types

```
TExeOffset = PtrUInt
```

The exact type of `TExeOffset` will differ per platform.

```
TExeProcessAddress = PtrUInt
```

The exact type of `TExeProcessAddress` will differ per platform.

50.4 Procedures and functions

50.4.1 CloseExeFile

Synopsis: Close a previously opened file.

Declaration: `function CloseExeFile(var e: TExeFile) : Boolean`

Visibility: default

Description: `CloseExeFile` closes an executable file image previously opened with OpenExeFile (744), and represented by `e`.

The function returns `True` if the file was closed successfully, or `False` if something went wrong.

Errors: In case of an error, `False` is returned.

See also: OpenExeFile (744)

50.4.2 FindExeSection

Synopsis: Find a section in the binary image.

Declaration:
```
function FindExeSection(var e: TExeFile; const secname: string;
                        var secofs: LongInt; var seclen: LongInt)
                         : Boolean
```

Visibility: default

Description: `FindExeSection` examines the binary that was opened with OpenExeFile (744) (represented by `e`) and searches for the section named `secname`. If found, the section offset is returned in `secofs` and the section length (in bytes) is returned in `seclen`.

The function returns `True` if the section was found, `False` if not.

See also: OpenExeFile (744)

50.4.3 GetModuleByAddr

Synopsis: Return the module name by address.

Declaration:
```
procedure GetModuleByAddr(addr: pointer; var baseaddr: pointer;
                          var filename: string)
```

Visibility: default

Description: `GetModuleByAddr` returns the name of the module that contains address `addr`. If successful, it returns `True` and returns the filename in `FileName` and the base address at which it is loaded in `BaseAddr`.

50.4.4 OpenExeFile

Synopsis: Open an executable file.

Declaration:
```
function OpenExeFile(var e: TExeFile; const fn: string) : Boolean
```

Visibility: default

Description: `OpenExeFile` opens the executable file `fn` and initializes the structure `e` for subsequent calls to routines in the `exeinfo` unit.

The function returns `True` if the file was opened successfully, false otherwise.

See also: FindExeSection (744), CloseExeFile (743), ReadDebugLink (745)

50.4.5 ReadDebugLink

Synopsis: Read the location of a debug info filename.

Declaration: `function ReadDebugLink(var e: TExeFile; var dbgfn: string) : Boolean`

Visibility: default

Description: `ReadDebugLink` examines the `.gnu_debuglink` section to see if the debug information is stored in an external file. If so, then the name of the file with the debug information is returned in the `dbgfn` parameter.

The function returns `false` if there is no external debug information file, or if the file with debug information does not exist. It is searched next to the binary file or in the current directory.

See also: OpenExeFile (744), CloseExeFile (743)

50.5 TExeFile

```
TExeFile = record
  f : File;
  size : Int64;
  isopen : Boolean;
  nsects : LongInt;
  sechdrofs : TExeOffset;
  secstrofs : TExeOffset
  ;
  processaddress : TExeProcessAddress;
  FunctionRelative : Boolean
  ;
  ImgOffset : TExeOffset;
  filename : string;
  buf : Array[0.
  .4095] of Byte;
  bufsize : LongInt;
  bufcnt : LongInt;
end
```

`TExeFile` is a record used in the various calls of this unit. It contains a file descriptor, and various fields that describe the executable.

The structure of `TExeFile` is opaque, that is, one shouldn't rely on the exactness of this structure, it may change any time in the future.

Chapter 51

Reference for unit 'fgl'

51.1 Used units

Table 51.1: Used units by unit 'fgl'

Name	Page
System	1362
sysutils	1635
Types	1965

51.2 Overview

The `fgl` unit contains some basic list-related generic classes.

51.3 Constants, types and variables

51.3.1 Constants

```
MaxGListSize = MaxInt div 1024deprecated
```

`MaxGListSize` is the maximum number of elements in the TFPGList (751) list.

```
MaxListSize = Maxint div 16
```

`MaxListSize` is the maximum number of elements a list can contain before the memory runs out.

51.3.2 Types

```
TFPSListCompareFunc = function(Key1: Pointer; Key2: Pointer) : Integer
  of object
```

`TFPSListCompareFunc` is used in the TFPSList.Sort (783) method to compare 2 elements. The list passes 2 pointers to the actual items to the compare function. The result of this function determines how the pointers will be sorted:

- If the result of this function is negative, the first key (`key1`) is assumed to be 'less' than the second key (`key2`) and will be moved before the second in the list.
- If the function result is positive, the first key (`key1`) pointer is assumed to be 'greater than' the second key (`key2`)and will be moved after the second in the list.
- if the function result is zero, the keys are assumed to be 'equal' and no moving will take place.

51.4 EListError

51.4.1 Description

`EListError` is the exception used in the TFPSList (777) class to indicate errors such as a list index out of bounds, wrong capacity etc.

See also: TFPSList.Capacity (783), TFPSList.Exchange (780), TFPSList.Items (784)

51.5 TFPGInterfacedObjectList

51.5.1 Description

`TFPGList` can be used to specialize a list for any class type `T` that requires reference counting (all objects that implement `IInterface` or `IUnknown`)). It will specialize to a list with the same methods as TFPSList (777) or classes.TFPList (746) or `TFPObjectList`

Classes that implement `IInterface` or `IUnknown` require special care to maintain the reference count. The `TFPGInterfacedObjectList` list provides the necessary functionality to deal with this.

See also: TFPSList (777), classes.TFPList (746)

51.5.2 Method overview

Page	Method	Description
748	Add	Add new object of class `T` to the list.
749	AddList	Adds the elements from another list.
749	Assign	Copy objects from Source list.
748	Create	Instantiate a new interfaced object list.
748	Extract	Extract an item from the list.
748	GetEnumerator	Return a list enumerator for `T`.
749	IndexOf	Index of object.
749	Insert	Insert a new object in the list.
750	Remove	Remove an object from the list.
750	Sort	Sort the objects in the list.

51.5.3 Property overview

Page	Properties	Access	Description
750	First	rw	First non-nil object.
751	Items	rw	Indexed access to objects in the list.
751	Last	rw	Last non-`Nil` object.
751	List	r	Internal list pointer.

51.5.4 TFPGInterfacedObjectList.Create

Synopsis: Instantiate a new interfaced object list.

Declaration: `constructor Create`

Visibility: public

Description: `Create` instantiates a new object list. It will simply call the inherited constructor with the correct item size.

See also: TFPSList.Destroy (778)

51.5.5 TFPGInterfacedObjectList.Add

Synopsis: Add new object of class `T` to the list.

Declaration: `function Add(const Item: T) : Integer`

Visibility: public

Description: `Add` adds a new item `Item` of class type `T` to the list and returns the position at which the item was added. Add will increase the reference count of the object.

Errors: If the item could not be added, an EListError (747) exception is raised.

See also: TFPGInterfacedObjectList.Extract (748), TFPGInterfacedObjectList.Items (751), TFPGInterfacedObjectList.IndexOf (749)

51.5.6 TFPGInterfacedObjectList.Extract

Synopsis: Extract an item from the list.

Declaration: `function Extract(const Item: T) : T`

Visibility: public

Description: `Extract` removes `Item` from the list and returns the removed item, or `Nil` if it was not found.

The extracted object will not be destroyed.

Errors: None.

See also: TFPSList.Delete (779)

51.5.7 TFPGInterfacedObjectList.GetEnumerator

Synopsis: Return a list enumerator for `T`.

Declaration: `function GetEnumerator : TFPGListEnumeratorSpec`

Visibility: public

Description: `GetEnumerator` returns an enumerator for the elements in the list. It is a specialized version of TFPGListEnumerator (756).

See also: TFPGListEnumerator (756)

51.5.8 TFPGInterfacedObjectList.IndexOf

Synopsis: Index of object.

Declaration: `function IndexOf(const Item: T) : Integer`

Visibility: public

Description: `IndexOf` returns the index of `Item` in the list, or -1 if the object does not appear in the list.

Errors: None.

See also: TFPGInterfacedObjectList.Items (751), TFPGInterfacedObjectList.Insert (749), TFPGInterfacedObjectList.Add (748)

51.5.9 TFPGInterfacedObjectList.Insert

Synopsis: Insert a new object in the list.

Declaration: `procedure Insert(Index: Integer; const Item: T)`

Visibility: public

Description: `Insert` inserts a new object (`Item`) in the list at position `Index`. The index is zero based and must be less than Count (784).

Errors: If an invalid index is specified, an EListError (747) exception is raised.

See also: TFPGInterfacedObjectList.Items (751), TFPGInterfacedObjectList.Add (748)

51.5.10 TFPGInterfacedObjectList.Assign

Synopsis: Copy objects from Source list.

Declaration: `procedure Assign(Source: TFPGInterfacedObjectList)`

Visibility: public

Description: `Assign` clears the list and copies all items in `Source` to the list. The source list must be of the same type as the destination list.

See also: TFPSList.Clear (779), TFPGObjectList.Add (773)

51.5.11 TFPGInterfacedObjectList.AddList

Synopsis: Adds the elements from another list.

Declaration: `procedure AddList(Source: TFPGInterfacedObjectList)`

Visibility: public

Description: `AddList` adds all the elements from list `Source` to the current list.

See also: TFPGList.AddList (754), TFPSList.AddList (782), TFPGObjectList.AddList (775), TFPGInterfacedObjectList.Add (748)

51.5.12 TFPGInterfacedObjectList.Remove

Synopsis: Remove an object from the list.

Declaration: `function Remove(const Item: T) : Integer`

Visibility: public

Description: `Remove` removes the object `Item` from the list, and returns the index of the removed item. If no item was removed, `-1` is returned. Only the first object is removed.

Removing an object from the list may cause the object to be freed.

Errors: None.

See also: TFPGInterfacedObjectList.IndexOf (749), TFPSList.Delete (779), TFPGInterfacedObjectList.FreeObjects (747)

51.5.13 TFPGInterfacedObjectList.Sort

Synopsis: Sort the objects in the list.

Declaration: `procedure Sort(Compare: TCompareFunc)`

Visibility: public

Description: `Sort` sorts the elements in the list using the provided `Compare` function. The list passes 2 items to the compare function. The result of this function determines how the items will be sorted:

- If the result of this function is negative, the first object (`Item1`) is assumed to be 'less' than the second object (`Item2`) and will be moved before the second in the list.
- If the function result is positive, the first object (`Item1`) is assumed to be 'greater than' the second object (`Item2`)and will be moved after the second in the list.
- if the function result is zero, the objects are assumed to be 'equal' and no moving will take place.

Errors: None.

See also: TFPSList.Sorted (777)

51.5.14 TFPGInterfacedObjectList.First

Synopsis: First non-nil object.

Declaration: `Property First : T`

Visibility: public

Access: Read,Write

Description: `First` returns the first non-nil object. If no such element is present, `Nil` is returned.

See also: TFPSList.First (785), TFPGInterfacedObjectList.Last (751), TFPSList.Pack (783)

51.5.15 TFPGInterfacedObjectList.Last

Synopsis: Last non-`Nil` object.

Declaration: `Property Last : T`

Visibility: public

Access: Read,Write

Description: `Last` returns the last non-`Nil` object. If no such element is present, `Nil` is returned.

See also: TFPGInterfacedObjectList.First (750), TFPSList.Last (785)

51.5.16 TFPGInterfacedObjectList.Items

Synopsis: Indexed access to objects in the list.

Declaration: `Property Items[Index: Integer]: T; default`

Visibility: public

Access: Read,Write

Description: `Items` provides indexed access to the objects in the list. The objects can be get or set.

The index `Index` is zero based, and has a maximum value of Count-1 (784).

The previous object at position `Index` may be freed when setting the property, depending on its reference count.

Errors: If an invalid index is used, an EListError (747) exception is raised.

See also: TFPSList.Count (784)

51.5.17 TFPGInterfacedObjectList.List

Synopsis: Internal list pointer.

Declaration: `Property List : PTypeList`

Visibility: public

Access: Read

Description: `List` is the internal list of objects. It should not be used directly.

See also: TFPGInterfacedObjectList.Items (751)

51.6 TFPGList

51.6.1 Description

`TFPGList` can be used to specialize a list for any type `T` that does not require reference counting (such as interfaced objects). It will specialize to a list with the same methods as TFPSList (777) or classes.TFPList (746)

See also: TFPSList (777), classes.TFPList (746)

51.6.2 Method overview

Page	Method	Description
752	Add	Add new item of type `T` to the list.
754	AddList	Adds the elements from another list.
754	Assign	Copy elements from Source list.
752	Create	Instantiate a new list.
753	Extract	Extract an item from the list.
753	GetEnumerator	Return a list enumerator for `T`.
753	IndexOf	Index of item.
753	Insert	Insert a new item in the list.
752	ItemIsManaged	Indicates whether the list item is a managed type or not.
754	Remove	Remove an item from the list.
754	Sort	Sort the list.

51.6.3 Property overview

Page	Properties	Access	Description
755	First	rw	First non-empty item.
755	Items	rw	Indexed access to items in the list.
755	Last	rw	Last non-empty item.
755	List	r	Internal list object.

51.6.4 TFPGList.Create

Synopsis: Instantiate a new list.

Declaration: `constructor Create`

Visibility: public

Description: `Create` instantiates a new list. It will simply call the inherited constructor with the correct item size: `sizeof(T)`.

51.6.5 TFPGList.ItemIsManaged

Synopsis: Indicates whether the list item is a managed type or not.

Declaration: `class function ItemIsManaged : Boolean;  Override`

Visibility: public

Description: `ItemIsManaged` indicates whether the list item type is a managed type or not. If not, faster codepaths are sometimes taken in the code.

51.6.6 TFPGList.Add

Synopsis: Add new item of type `T` to the list.

Declaration: `function Add(const Item: T) : Integer`

Visibility: public

Description: `Add` adds a new item `Item` of generic type `T` to the list and returns the position at which the item was added.

Errors: If the item could not be added, an EListError (747) exception is raised.

See also: TFPGList.Extract (753), TFPGList.Items (755), TFPGList.IndexOf (753)

51.6.7 TFPGList.Extract

Synopsis: Extract an item from the list.

Declaration: `function Extract(const Item: T) : T`

Visibility: public

Description: `Extract` removes `Item` from the list and returns the removed item, or an expression equivalent to `T(0)` if it was not found.

Errors: None.

See also: TFPSList.Delete (779)

51.6.8 TFPGList.GetEnumerator

Synopsis: Return a list enumerator for `T`.

Declaration: `function GetEnumerator : TFPGListEnumeratorSpec`

Visibility: public

Description: `GetEnumerator` returns an enumerator for the elements in the list. It is a specialized version of TFPGListEnumerator (756).

See also: TFPGListEnumerator (756)

51.6.9 TFPGList.IndexOf

Synopsis: Index of item.

Declaration: `function IndexOf(const Item: T) : Integer`

Visibility: public

Description: `IndexOf` returns the index of `Item` in the list, or -1 if the item does not appear in the list.

Errors: None.

See also: TFPGList.Items (755), TFPGList.Insert (753), TFPSList.Add (779)

51.6.10 TFPGList.Insert

Synopsis: Insert a new item in the list.

Declaration: `procedure Insert(Index: Integer; const Item: T)`

Visibility: public

Description: `Insert` inserts a new `item` in the list at position `Index`. The index is zero based and must be less than Count (784).

Errors: If an invalid index is specified, an EListError (747) exception is raised.

See also: TFPGList.Items (755), TFPGList.Insert (753), TFPSList.Add (779)

51.6.11 TFPGList.Assign

Synopsis: Copy elements from Source list.

Declaration: `procedure Assign(Source: TFPGList)`

Visibility: public

Description: `Assign` clears the list and copies all items in `Source` to the list. The source list must be of the same type as the destination list.

See also: TFPSList.Clear (779), TFPGList.Add (752)

51.6.12 TFPGList.AddList

Synopsis: Adds the elements from another list.

Declaration: `procedure AddList(Source: TFPGList)`

Visibility: public

Description: `AddList` adds all the elements from list `Source` to the current list.

See also: TFPGList.Add (752), TFPSList.AddList (782), TFPGObjectList.AddList (775)

51.6.13 TFPGList.Remove

Synopsis: Remove an item from the list.

Declaration: `function Remove(const Item: T) : Integer`

Visibility: public

Description: `Remove` removes the item `Item` from the list, and returns the index of the removed item. If no item was removed, `-1` is returned. Only the first item is removed.

Errors: None.

See also: TFPGList.IndexOf (753)

51.6.14 TFPGList.Sort

Synopsis: Sort the list.

Declaration: `procedure Sort(Compare: TCompareFunc)`

Visibility: public

Description: `Sort` sorts the elements in the list using the provided `Compare` function. The list passes 2 items to the compare function. The result of this function determines how the items will be sorted:

- If the result of this function is negative, the first item (`Item1`) is assumed to be 'less' than the second item (`Item2`) and will be moved before the second in the list.
- If the function result is positive, the first item (`Item1`) is assumed to be 'greater than' the second item (`Item2`)and will be moved after the second in the list.
- if the function result is zero, the items are assumed to be 'equal' and no moving will take place.

51.6.15 TFPGList.First

Synopsis: First non-empty item.

Declaration: `Property First : T`

Visibility: public

Access: Read,Write

Description: `First` returns the first non-empty item, which means the first item not equal to `T(0)`. If no such element is present, `T(0)`. is returned.

See also: TFPSList.First (785), TFPGList.Last (755)

51.6.16 TFPGList.Last

Synopsis: Last non-empty item.

Declaration: `Property Last : T`

Visibility: public

Access: Read,Write

Description: `Last` returns the last non-empty item, which means the last item not equal to `T(0)`. If no such element is present, `T(0)`. is returned.

See also: TFPGList.First (755), TFPSList.Last (785)

51.6.17 TFPGList.Items

Synopsis: Indexed access to items in the list.

Declaration: `Property Items[Index: Integer]: T; default`

Visibility: public

Access: Read,Write

Description: `Items` provides indexed access to the items in the list. The items can be get or set.

The index `Index` is zero based, and has a maximum value of Count-1 (784).

Errors: If an invalid index is used, an EListError (747) exception is raised.

See also: TFPSList.Count (784)

51.6.18 TFPGList.List

Synopsis: Internal list object.

Declaration: `Property List : PTypeList`

Visibility: public

Access: Read

Description: `List` is the internal list of items. It should not be used directly.

See also: TFPGList.Items (755)

51.7 TFPGListEnumerator

51.7.1 Description

`TFPGListEnumerator` is a generic list enumerator. It is used in the TFPGList (751) class to implement the enumerator for the list.

Normally there should be no need to instantiate or use this class directly.

See also: TFPGList (751)

51.7.2 Method overview

Page	Method	Description
756	Create	Create a new list enumerator.
756	MoveNext	Move to next element in the list.

51.7.3 Property overview

Page	Properties	Access	Description
756	Current	r	Current enumerated element.

51.7.4 TFPGListEnumerator.Create

Synopsis: Create a new list enumerator.

Declaration: `constructor Create(AList: TFPSList)`

Visibility: public

Description: `Create` is called by the list `AList` to initialize a new enumerator. There should be no need to call this directly.

See also: TFPGList (751)

51.7.5 TFPGListEnumerator.MoveNext

Synopsis: Move to next element in the list.

Declaration: `function MoveNext : Boolean`

Visibility: public

Description: `MoveNext` moves to the next element in the list.

See also: TFPGListEnumerator.Current (756)

51.7.6 TFPGListEnumerator.Current

Synopsis: Current enumerated element.

Declaration: `Property Current : T`

Visibility: public

Access: Read

Description: `Current` returns the currently enumerated element. It is only valid after TFPGListEnumerator.MoveNext (756) was called and returned `True`.

See also: TFPGListEnumerator.MoveNext (756)

51.8 TFPGMap

51.8.1 Description

`TFPGMap` is a generic map class. It can be used to specialize a map for any key type and data type that do not require manual reference counting: For reference counted interface objects, TFPGMapInterfacedObjectData (762) must be used.

See also: TFPGMapInterfacedObjectData (762)

51.8.2 Method overview

Page	Method	Description
758	Add	Add a key and value to the map.
758	AddOrSetData	Add data with given or set value if the key already exists.
757	Create	Create a new instance of the map.
758	Find	Find item based on key.
759	IndexOf	Find index of a key in the list.
759	IndexOfData	Find index of data value in the list.
759	InsertKey	Insert a new key in the list.
759	InsertKeyData	Insert a new key with associated data in the list.
760	Remove	Remove a key from the list.
758	TryGetData	Find data or return default.

51.8.3 Property overview

Page	Properties	Access	Description
760	Data	rw	Indexed access to the data in the list.
761	KeyData	rw	Access to data based on key.
760	Keys	rw	Indexed access to the keys in the list.
761	OnCompare	rw	Alias for `OnKeyCompare`.
762	OnDataCompare	rw	Compare function for data values.
761	OnKeyCompare	rw	Compare function for key values.

51.8.4 TFPGMap.Create

Synopsis: Create a new instance of the map.

Declaration: `constructor Create`

Visibility: public

Description: `Create` instantiates a new map. It mainly initializes the TFPSMap (785) parent with the sized of the key and data.

See also: TFPSMap.Create (786)

51.8.5 TFPGMap.Add

Synopsis: Add a key and value to the map.

Declaration:
```
function Add(const AKey: TKey; const AData: TData) : Integer
function Add(const AKey: TKey) : Integer
```

Visibility: public

Description: `Add` adds a new key `AKey` of generic type `TKey` with data value `AData` to the list and returns the position at which the key was added.

Errors: If the item could not be added, an EListError (747) exception is raised. If Duplicates (789) is set to `dupError` and a duplicate key is added, an EListError (747) exception is raised.

See also: TFPGMap.Keys (760), TFPGMap.IndexOf (759), TFPGMap.KeyData (761), TFPGMap.Data (760), TFPSMap.Duplicates (789)

51.8.6 TFPGMap.Find

Synopsis: Find item based on key.

Declaration:
```
function Find(const AKey: TKey; out Index: Integer) : Boolean
```

Visibility: public

Description: `Find` will search the key equal to `AKey` and return its index in `AIndex`. The return value of the function is `True` if an exact match for `AKey` is found, `False` otherwise.

The behaviour of Find is undefined if the map is not sorted. For unsorted maps, use IndexOf (759) instead.

See also: TFPGMap.IndexOf (759), TFPGMap.IndexOfData (759)

51.8.7 TFPGMap.TryGetData

Synopsis: Find data or return default.

Declaration:
```
function TryGetData(const AKey: TKey; out AData: TData) : Boolean
```

Visibility: public

Description: `TryGetData` will search the map for `AKey` and return `True` or `False` depending on whether the value with the given key was found. If the key was found, the associated value is returned in `AData`, if it is not found a default value (using `Default`) is returned.

Errors: None.

See also: TFPGMap.Find (758)

51.8.8 TFPGMap.AddOrSetData

Synopsis: Add data with given or set value if the key already exists.

Declaration:
```
procedure AddOrSetData(const AKey: TKey; const AData: TData)
```

Visibility: public

Description: `AddOrSetData` will check if key `AKey` already exists. if yes, the value associated with it will be replaced with `AData`. If the key does not yet exist, it will be added with value `AData`.

Errors: None.

See also: TFPGMap.TryGetData (758), TFPGMap.Add (758), TFPGMap.Find (758)

51.8.9 TFPGMap.IndexOf

Synopsis: Find index of a key in the list.

Declaration:
```
function IndexOf(const AKey: TKey) : Integer
```

Visibility: public

Description: `IndexOf` returns the index of `AKey` in the list, or -1 if the key was not found in the list.

Errors: None.

See also: TFPGMap.Find (758), TFPGMap.IndexOfData (759)

51.8.10 TFPGMap.IndexOfData

Synopsis: Find index of data value in the list.

Declaration:
```
function IndexOfData(const AData: TData) : Integer
```

Visibility: public

Description: `IndexOfData` returns the index of `AData` in the list, or -1 if the data was not found in the list.

Errors: None.

See also: TFPGMap.Find (758), TFPGMap.IndexOf (759)

51.8.11 TFPGMap.InsertKey

Synopsis: Insert a new key in the list.

Declaration:
```
procedure InsertKey(Index: Integer; const AKey: TKey)
```

Visibility: public

Description: `InsertKey` inserts key `AKey` at position `Index` in the list. It is not allowed to insert a key in a sorted list.

Errors: If the index `AIndex` is out of range [0..Count-1], or the list is sorted, an EListError (747) exception will be raised.

See also: TFPGMap.InsertKeyData (759), TFPGMap.Add (758), TFPSMap.Delete (785), TFPSMap.Remove (788)

51.8.12 TFPGMap.InsertKeyData

Synopsis: Insert a new key with associated data in the list.

Declaration:
```
procedure InsertKeyData(Index: Integer; const AKey: TKey;
                        const AData: TData)
```

Visibility: public

Description: `InsertKey` inserts key `AKey` with associated data `AData` at position `Index` in the list. It is not allowed to insert a key in a sorted list.

Errors: If the index `AIndex` is out of range [0..Count-1], or the list is sorted, an EListError (747) exception will be raised.

See also: TFPGMap.InsertKey (759), TFPGMap.Add (758), TFPSMap.Delete (785), TFPGMap.Remove (760)

51.8.13 TFPGMap.Remove

Synopsis: Remove a key from the list.

Declaration: `function Remove(const AKey: TKey) : Integer`

Visibility: public

Description: `Remove` removes the key `AKey` from the list, together with its associated data. The function returns the index of `AKey` prior to removal from the list, or -1 if `AKey` was not present in the list.

Errors: None.

See also: TFPGMap.InsertKey (759), TFPGMap.InsertKeyData (759), TFPGMap.Add (758), TFPSMap.Delete (785)

51.8.14 TFPGMap.Keys

Synopsis: Indexed access to the keys in the list.

Declaration: `Property Keys[Index: Integer]: TKey`

Visibility: public

Access: Read,Write

Description: `Keys` provides indexed access to the key values in the list. Valid values for `Index` are in the range `[0..Count-1]`. Key values can always be read, but can only be written if the list is unsorted.

Errors: If the index `AIndex` is out of range [0..Count-1], an EListError (747) exception will be raised. The same exception is raised if a key is written and the list is sorted.

See also: TFPSList.Count (784), TFPGMap.Data (760), TFPGMap.KeyData (761)

51.8.15 TFPGMap.Data

Synopsis: Indexed access to the data in the list.

Declaration: `Property Data[Index: Integer]: TData`

Visibility: public

Access: Read,Write

Description: `Data` provides indexed access to the data values in the list. Valid values for `Index` are in the range `[0..Count-1]`. Data can always be read or written.

Errors: If the index `AIndex` is out of range [0..Count-1], an EListError (747) exception will be raised.

See also: TFPSList.Count (784), TFPGMap.Keys (760), TFPGMap.KeyData (761)

51.8.16 TFPGMap.KeyData

Synopsis: Access to data based on key.

Declaration: `Property KeyData[AKey: TKey]: TData; default`

Visibility: public

Access: Read,Write

Description: `KeyData` allows access to the data based on the key value `AKey`. The data can be read and written. When writing, writing using an existing key will overwrite the current data. If it does not exist yet, it will be created. When reading, if the key is not present, an EListError (747) will be raised.

Errors: If the key does not exist, an EListError (747) exception will be raised.

See also: TFPSList.Count (784), TFPGMap.Keys (760), TFPGMap.Data (760)

51.8.17 TFPGMap.OnCompare

Synopsis: Alias for `OnKeyCompare`.

Declaration: `Property OnCompare : TKeyCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnCompare` is a deprecated property, use TFPGMap.OnKeyCompare (761) instead.

See also: TFPGMap.OnKeyCompare (761)

51.8.18 TFPGMap.OnKeyCompare

Synopsis: Compare function for key values.

Declaration: `Property OnKeyCompare : TKeyCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnKeyCompare` can be set to a function that compares key values. The default value for this event is a function that compares keys based on a byte-by-byte comparison of the memory block. The function must have the following semantics:

- If the result of this function is negative, the first key (`key1`) is assumed to be 'less' than the second key (`key2`) and will be moved before the second in the list.
- If the function result is positive, the first key (`key1`) pointer is assumed to be 'greater than' the second key (`key2`)and will be moved after the second in the list.
- if the function result is zero, the keys are assumed to be 'equal' and no moving will take place.

See also: TFPGMap.OnDataCompare (762)

51.8.19 TFPGMap.OnDataCompare

Synopsis: Compare function for data values.

Declaration: `Property OnDataCompare : TDataCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnDataCompare` can be set to a function that compares data values. The default value for this event is a function that compares data based on a byte-by-byte comparison of the memory block. The function must have the following semantics:

- If the result of this function is negative, the first data item (`Data1`) is assumed to be 'less' than the second data item (`Data2`) and will be moved before the second in the list.
- If the function result is positive, the first data item (`Data1`) pointer is assumed to be 'greater than' the second data item (`Data2`)and will be moved after the second in the list.
- if the function result is zero, the data items are assumed to be 'equal' and no moving will take place.

See also: TFPGMap.OnKeyCompare (761)

51.9 TFPGMapInterfacedObjectData

51.9.1 Description

`TFPGInterfacedObjectMap` is a generic map class. It can be used to specialize a map for any key type, with associated data type that requires manual reference counting: any type which implements `IInterface`. For non-reference counted objects, TFPGMap (757) should be used.

This map class is entirely equivalent to TFPGMap (757), but operates on data items that require additional reference counting code on the data.

See also: TFPGMap (757)

51.9.2 Method overview

Page	Method	Description
763	Add	Add a key and value to the map.
764	AddOrSetData	Add data with given or set value if the key already exists.
763	Create	Create a new instance of the map.
763	Find	Find item based on key.
764	IndexOf	Find index of a key in the list.
764	IndexOfData	Find index of data value in the list.
765	InsertKey	Insert a new key in the list.
765	InsertKeyData	Insert a new key with associated data in the list.
765	Remove	Remove a key from the list.
764	TryGetData	Find data or return default.

51.9.3 Property overview

Page	Properties	Access	Description
766	Data	rw	Indexed access to the data in the list.
766	KeyData	rw	Access to data based on key.
765	Keys	rw	Indexed access to the keys in the list.
766	OnCompare	rw	Alias for `OnKeyCompare`.
767	OnDataCompare	rw	Compare function for data values.
767	OnKeyCompare	rw	Compare function for key values.

51.9.4 TFPGMapInterfacedObjectData.Create

Synopsis: Create a new instance of the map.

Declaration:
```
constructor Create
```

Visibility: public

Description: `Create` instantiates a new map. It mainly initializes the TFPSMap (785) parent with the sized of the key and data.

See also: TFPSMap.Create (786)

51.9.5 TFPGMapInterfacedObjectData.Add

Synopsis: Add a key and value to the map.

Declaration:
```
function Add(const AKey: TKey; const AData: TData) : Integer
function Add(const AKey: TKey) : Integer
```

Visibility: public

Description: `Add` adds a new key `AKey` of generic type `TKey` with data value `AData` to the list and returns the position at which the key was added.

Errors: If the item could not be added, an EListError (747) exception is raised. If Duplicates (789) is set to `dupError` and a duplicate key is added, an EListError (747) exception is raised.

See also: TFPGMapInterfacedObjectData.Keys (765), TFPGMapInterfacedObjectData.IndexOf (764), TFPGMapInterfacedObjectData.KeyData (766), TFPGMapInterfacedObjectData.Data (766), TFPS.Duplicates (746)

51.9.6 TFPGMapInterfacedObjectData.Find

Synopsis: Find item based on key.

Declaration:
```
function Find(const AKey: TKey; out Index: Integer) : Boolean
```

Visibility: public

Description: `Find` will search the first key smaller than or equal to `AKey` and return its index in `AIndex`. If the key was not found then -1 is returned. The return value of the function is `True` if an exact match for `AKey` is found, `False` otherwise.

See also: TFPGMapInterfacedObjectData.IndexOf (764), TFPGMapInterfacedObjectData.IndexOfData (764)

51.9.7 TFPGMapInterfacedObjectData.TryGetData

Synopsis: Find data or return default.

Declaration: `function TryGetData(const AKey: TKey; out AData: TData) : Boolean`

Visibility: public

Description: `TryGetData` will search the map for `AKey` and return `True` or `False` depending on whether the value with the given key was found. If the key was found, the associated value is returned in `AData`, if it is not found a default value (using `Default`) is returned.

Errors: None.

See also: TFPGMapInterfacedObjectData.Find (763)

51.9.8 TFPGMapInterfacedObjectData.AddOrSetData

Synopsis: Add data with given or set value if the key already exists.

Declaration: `procedure AddOrSetData(const AKey: TKey; const AData: TData)`

Visibility: public

Description: `AddOrSetData` will check if key `AKey` already exists. if yes, the value associated with it will be replaced with `AData`. If the key does not yet exist, it will be added with value `AData`.

Errors: None.

See also: TFPGMapInterfacedObjectData.TryGetData (764), TFPGMapInterfacedObjectData.Add (763), TFPGMapInterfacedObjectData.Find (763)

51.9.9 TFPGMapInterfacedObjectData.IndexOf

Synopsis: Find index of a key in the list.

Declaration: `function IndexOf(const AKey: TKey) : Integer`

Visibility: public

Description: `IndexOf` returns the index of `AKey` in the list, or -1 if the key was not found in the list.

Errors: None.

See also: TFPGMapInterfacedObjectData.Find (763), TFPGMapInterfacedObjectData.IndexOfData (764)

51.9.10 TFPGMapInterfacedObjectData.IndexOfData

Synopsis: Find index of data value in the list.

Declaration: `function IndexOfData(const AData: TData) : Integer`

Visibility: public

Description: `IndexOfData` returns the index of `AData` in the list, or -1 if the data was not found in the list.

Errors: None.

See also: TFPGMapInterfacedObjectData.Find (763), TFPGMapInterfacedObjectData.IndexOf (764)

51.9.11 TFPGMapInterfacedObjectData.InsertKey

Synopsis: Insert a new key in the list.

Declaration: `procedure InsertKey(Index: Integer; const AKey: TKey)`

Visibility: public

Description: `InsertKey` inserts key `AKey` at position `Index` in the list. It is not allowed to insert a key in a sorted list.

Errors: If the index `AIndex` is out of range [0..Count-1], or the list is sorted, an EListError (747) exception will be raised.

See also: TFPGMapInterfacedObjectData.InsertKeyData (765), TFPGMapInterfacedObjectData.Add (763), TFPSMapInterfacedObjectData.Delete (746), TFPSMapInterfacedObjectData.Remove (746)

51.9.12 TFPGMapInterfacedObjectData.InsertKeyData

Synopsis: Insert a new key with associated data in the list.

Declaration:
```
procedure InsertKeyData(Index: Integer; const AKey: TKey;
                        const AData: TData)
```

Visibility: public

Description: `InsertKey` inserts key `AKey` with associated data `AData` at position `Index` in the list. It is not allowed to insert a key in a sorted list.

Errors: If the index `AIndex` is out of range [0..Count-1], or the list is sorted, an EListError (747) exception will be raised.

See also: TFPGMapInterfacedObjectData.InsertKey (765), TFPGMapInterfacedObjectData.Add (763), TFPSMapInterfacedObjectData.Delete (746), TFPGMapInterfacedObjectData.Remove (765)

51.9.13 TFPGMapInterfacedObjectData.Remove

Synopsis: Remove a key from the list.

Declaration: `function Remove(const AKey: TKey) : Integer`

Visibility: public

Description: `Remove` removes the key `AKey` from the list, together with its associated data. The function returns the index of `AKey` prior to removal from the list, or -1 if `AKey` was not present in the list.

Errors: None.

See also: TFPGMap.InsertKey (759), TFPGMap.InsertKeyData (759), TFPGMap.Add (758), TFPGMapInterfacedObjectData.Delete (762)

51.9.14 TFPGMapInterfacedObjectData.Keys

Synopsis: Indexed access to the keys in the list.

Declaration: `Property Keys[Index: Integer]: TKey`

Visibility: public

Access: Read,Write

Description: `Keys` provides indexed access to the key values in the list. Valid values for `Index` are in the range `[0..Count-1]`. Key values can always be read, but can only be written if the list is unsorted.

Errors: If the index `AIndex` is out of range [0..Count-1], an EListError (747) exception will be raised. The same exception is raised if a key is written and the list is sorted.

See also: TFPSList.Count (784), TFPGMapInterfacedObjectData.Data (766), TFPGMapInterfacedObjectData.KeyData (766)

51.9.15 TFPGMapInterfacedObjectData.Data

Synopsis: Indexed access to the data in the list.

Declaration: `Property Data[Index: Integer]: TData`

Visibility: public

Access: Read,Write

Description: `Data` provides indexed access to the data values in the list. Valid values for `Index` are in the range `[0..Count-1]`. Data can always be read or written.

Errors: If the index `AIndex` is out of range [0..Count-1], an EListError (747) exception will be raised.

See also: TFPSList.Count (784), TFPGMapInterfacedObjectData.Keys (765), TFPGMapInterfacedObjectData.KeyData (766)

51.9.16 TFPGMapInterfacedObjectData.KeyData

Synopsis: Access to data based on key.

Declaration: `Property KeyData[AKey: TKey]: TData; default`

Visibility: public

Access: Read,Write

Description: `KeyData` allows access to the data based on the key value `AKey`. The data can be read and written. When writing, writing using an existing key will overwrite the current data. If it does not exist yet, it will be created. When reading, if the key is not present, an EListError (747) will be raised.

Errors: If the key does not exist, an EListError (747) exception will be raised.

See also: TFPSList.Count (784), TFPGMapInterfacedObjectData.Keys (765), TFPGMapInterfacedObjectData.Data (766)

51.9.17 TFPGMapInterfacedObjectData.OnCompare

Synopsis: Alias for `OnKeyCompare`.

Declaration: `Property OnCompare : TKeyCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnCompare` is a deprecated property, use TFPGMapInterfacedObjectData.OnKeyCompare (767) instead.

See also: TFPGMapInterfacedObjectData.OnKeyCompare (767)

51.9.18 TFPGMapInterfacedObjectData.OnKeyCompare

Synopsis: Compare function for key values.

Declaration: `Property OnKeyCompare : TKeyCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnKeyCompare` can be set to a function that compares key values. The default value for this event is a function that compares keys based on a byte-by-byte comparison of the memory block. The function must have the following semantics:

- If the result of this function is negative, the first key (`key1`) is assumed to be 'less' than the second key (`key2`) and will be moved before the second in the list.
- If the function result is positive, the first key (`key1`) pointer is assumed to be 'greater than' the second key (`key2`)and will be moved after the second in the list.
- if the function result is zero, the keys are assumed to be 'equal' and no moving will take place.

See also: TFPGMapInterfacedObjectData.OnDataCompare (767)

51.9.19 TFPGMapInterfacedObjectData.OnDataCompare

Synopsis: Compare function for data values.

Declaration: `Property OnDataCompare : TDataCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnDataCompare` can be set to a function that compares data values. The default value for this event is a function that compares data based on a byte-by-byte comparison of the memory block. The function must have the following semantics:

- If the result of this function is negative, the first data item (`Data1`) is assumed to be 'less' than the second data item (`Data2`) and will be moved before the second in the list.
- If the function result is positive, the first data item (`Data1`) pointer is assumed to be 'greater than' the second data item (`Data2`)and will be moved after the second in the list.
- if the function result is zero, the data items are assumed to be 'equal' and no moving will take place.

See also: TFPGMapInterfacedObjectData.OnKeyCompare (767)

51.10 TFPGMapObject

51.10.1 Description

`TFPGMapObject` is a generic map class. It can be used to specialize a map for object types, but not objects with automated reference counting: For reference counted interface objects, TFPGMapInterfacedObjectData (762) must be used.

See also: TFPGMapInterfacedObjectData (762)

51.10.2 Method overview

Page	Method	Description
768	Add	Add a key and value to the map.
769	AddOrSetData	Add data with given or set value if the key already exists.
768	Create	Create a new instance of the map.
769	Find	Find item based on key.
769	IndexOf	Find index of a key in the list.
770	IndexOfData	Find index of data value in the list.
770	InsertKey	Insert a new key in the list.
770	InsertKeyData	Insert a new key with associated data in the list.
770	Remove	Remove a key from the list.
769	TryGetData	Find data or return default.

51.10.3 Property overview

Page	Properties	Access	Description
771	Data	rw	Indexed access to the data in the list.
771	KeyData	rw	Access to data based on key.
771	Keys	rw	Indexed access to the keys in the list.
772	OnCompare	rw	Alias for `OnKeyCompare`.
772	OnDataCompare	rw	Compare function for data values.
772	OnKeyCompare	rw	Compare function for key values.

51.10.4 TFPGMapObject.Create

Synopsis: Create a new instance of the map.

Declaration:
```
constructor Create(AFreeObjects: Boolean)
constructor Create
```

Visibility: public

Description: `Create` instantiates a new map. It mainly initializes the TFPSMap (785) parent with the sized of the key and data.

See also: TFPSMap.Create (786)

51.10.5 TFPGMapObject.Add

Synopsis: Add a key and value to the map.

Declaration:
```
function Add(const AKey: TKey; const AData: TData) : Integer
function Add(const AKey: TKey) : Integer
```

Visibility: public

Description: `Add` adds a new key `AKey` of generic type `TKey` with data object `AData` to the list and returns the position at which the key was added.

Errors: If the item could not be added, an EListError (747) exception is raised. If Duplicates (789) is set to `dupError` and a duplicate key is added, an EListError (747) exception is raised.

See also: TFPGMapObject.Keys (771), TFPGMapObject.IndexOf (769), TFPGMapObject.KeyData (771), TFPGMapObject.Data (771), TFPSMap.Duplicates (789)

51.10.6 TFPGMapObject.Find

Synopsis: Find item based on key.

Declaration: `function Find(const AKey: TKey; out Index: Integer) : Boolean`

Visibility: public

Description: `Find` will search the first key smaller than or equal to `AKey` and return its index in `AIndex`. If the key was not found then -1 is returned. The return value of the function is `True` if an exact match for `AKey` is found, `False` otherwise.

See also: TFPGMapObject.IndexOf (769), TFPGMapObject.IndexOfData (770)

51.10.7 TFPGMapObject.TryGetData

Synopsis: Find data or return default.

Declaration: `function TryGetData(const AKey: TKey; out AData: TData) : Boolean`

Visibility: public

Description: `TryGetData` will search the map for `AKey` and return `True` or `False` depending on whether the value with the given key was found. If the key was found, the associated value is returned in `AData`, if it is not found a default value (using `Default`) is returned.

Errors: None.

See also: TFPGMapObject.Find (769)

51.10.8 TFPGMapObject.AddOrSetData

Synopsis: Add data with given or set value if the key already exists.

Declaration: `procedure AddOrSetData(const AKey: TKey; const AData: TData)`

Visibility: public

Description: `AddOrSetData` will check if key `AKey` already exists. if yes, the value associated with it will be replaced with `AData`. If the key does not yet exist, it will be added with value `AData`.

Errors: None.

See also: TFPGMapObject.TryGetData (769), TFPGMapObject.Add (768), TFPGMapObject.Find (769)

51.10.9 TFPGMapObject.IndexOf

Synopsis: Find index of a key in the list.

Declaration: `function IndexOf(const AKey: TKey) : Integer`

Visibility: public

Description: `IndexOf` returns the index of `AKey` in the list, or -1 if the key was not found in the list.

Errors: None.

See also: TFPGMapObject.Find (769), TFPGMapObject.IndexOfData (770)

51.10.10 TFPGMapObject.IndexOfData

Synopsis: Find index of data value in the list.

Declaration:
```
function IndexOfData(const AData: TData) : Integer
```

Visibility: public

Description: `IndexOfData` returns the index of object `AData` in the list, or -1 if the data was not found in the list.

Errors: None.

See also: TFPGMapObject.Find (769), TFPGMapObject.IndexOf (769)

51.10.11 TFPGMapObject.InsertKey

Synopsis: Insert a new key in the list.

Declaration:
```
procedure InsertKey(Index: Integer; const AKey: TKey)
```

Visibility: public

Description: `InsertKey` inserts key `AKey` at position `Index` in the list. It is not allowed to insert a key in a sorted list.

Errors: If the index `AIndex` is out of range [0..Count-1], or the list is sorted, an EListError (747) exception will be raised.

See also: TFPGMapObject.InsertKeyData (770), TFPGMapObject.Add (768), TFPSMap.Delete (785), TFPSMap.Remove (788)

51.10.12 TFPGMapObject.InsertKeyData

Synopsis: Insert a new key with associated data in the list.

Declaration:
```
procedure InsertKeyData(Index: Integer; const AKey: TKey;
                        const AData: TData)
```

Visibility: public

Description: `InsertKey` inserts key `AKey` with associated data `AData` at position `Index` in the list. It is not allowed to insert a key in a sorted list.

Errors: If the index `AIndex` is out of range [0..Count-1], or the list is sorted, an EListError (747) exception will be raised.

See also: TFPGMapObject.InsertKey (770), TFPGMapObject.Add (768), TFPSMap.Delete (785), TFPGMapObject.Remove (770)

51.10.13 TFPGMapObject.Remove

Synopsis: Remove a key from the list.

Declaration:
```
function Remove(const AKey: TKey) : Integer
```

Visibility: public

Description: `Remove` removes the key `AKey` from the list, together with its associated data. The function returns the index of `AKey` prior to removal from the list, or -1 if `AKey` was not present in the list.

Errors: None.

See also: TFPGMapObject.InsertKey (770), TFPGMapObject.InsertKeyData (770), TFPGMapObject.Add (768), TFPSMap.Delete (785)

51.10.14 TFPGMapObject.Keys

Synopsis: Indexed access to the keys in the list.

Declaration: `Property Keys[Index: Integer]: TKey`

Visibility: public

Access: Read,Write

Description: `Keys` provides indexed access to the key values in the list. Valid values for `Index` are in the range `[0..Count-1]`. Key values can always be read, but can only be written if the list is unsorted.

Errors: If the index `AIndex` is out of range [0..Count-1], an EListError (747) exception will be raised. The same exception is raised if a key is written and the list is sorted.

See also: TFPSList.Count (784), TFPGMapObject.Data (771), TFPGMapObject.KeyData (771)

51.10.15 TFPGMapObject.Data

Synopsis: Indexed access to the data in the list.

Declaration: `Property Data[Index: Integer]: TData`

Visibility: public

Access: Read,Write

Description: `Data` provides indexed access to the data values in the list. Valid values for `Index` are in the range `[0..Count-1]`. Data can always be read or written.

Errors: If the index `AIndex` is out of range [0..Count-1], an EListError (747) exception will be raised.

See also: TFPSList.Count (784), TFPGMapObject.Keys (771), TFPGMapObject.KeyData (771)

51.10.16 TFPGMapObject.KeyData

Synopsis: Access to data based on key.

Declaration: `Property KeyData[AKey: TKey]: TData; default`

Visibility: public

Access: Read,Write

Description: `KeyData` allows access to the data based on the key value `AKey`. The data can be read and written. When writing, writing using an existing key will overwrite the current data. If it does not exist yet, it will be created. When reading, if the key is not present, an EListError (747) will be raised.

Errors: If the key does not exist, an EListError (747) exception will be raised.

See also: TFPSList.Count (784), TFPGMapObject.Keys (771), TFPGMapObject.Data (771)

51.10.17 TFPGMapObject.OnCompare

Synopsis: Alias for `OnKeyCompare`.

Declaration: `Property OnCompare : TKeyCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnCompare` is a deprecated property, use TFPGMapObject.OnKeyCompare (772) instead.

See also: TFPGMapObject.OnKeyCompare (772)

51.10.18 TFPGMapObject.OnKeyCompare

Synopsis: Compare function for key values.

Declaration: `Property OnKeyCompare : TKeyCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnKeyCompare` can be set to a function that compares key values. The default value for this event is a function that compares keys based on a byte-by-byte comparison of the memory block. The function must have the following semantics:

- If the result of this function is negative, the first key (`key1`) is assumed to be 'less' than the second key (`key2`) and will be moved before the second in the list.
- If the function result is positive, the first key (`key1`) pointer is assumed to be 'greater than' the second key (`key2`)and will be moved after the second in the list.
- if the function result is zero, the keys are assumed to be 'equal' and no moving will take place.

See also: TFPGMapObject.OnDataCompare (772)

51.10.19 TFPGMapObject.OnDataCompare

Synopsis: Compare function for data values.

Declaration: `Property OnDataCompare : TDataCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnDataCompare` can be set to a function that compares data values. The default value for this event is a function that compares data based on a byte-by-byte comparison of the memory block. The function must have the following semantics:

- If the result of this function is negative, the first data item (`Data1`) is assumed to be 'less' than the second data item (`Data2`) and will be moved before the second in the list.
- If the function result is positive, the first data item (`Data1`) pointer is assumed to be 'greater than' the second data item (`Data2`)and will be moved after the second in the list.
- if the function result is zero, the data items are assumed to be 'equal' and no moving will take place.

See also: TFPGMapObject.OnKeyCompare (772)

51.11 TFPGObjectList

51.11.1 Description

`TFPGList` can be used to specialize a list for any class type `T` that does not require reference counting (such as interfaced objects). It will specialize to a list with the same methods as TFPSList (777) or classes.TFPList (746) or `TFPObjectList`

See also: TFPSList (777), classes.TFPList (746)

51.11.2 Method overview

Page	Method	Description
773	Add	Add new object of class `T` to the list.
775	AddList	Adds the elements from another list.
775	Assign	Copy objects from Source list.
773	Create	Instantiate a new object list.
774	Extract	Extract an item from the list.
774	GetEnumerator	Return a list enumerator for `T`.
774	IndexOf	Index of item.
775	Insert	Insert a new object in the list.
775	Remove	Remove an object from the list.
776	Sort	Sort the objects in the list.

51.11.3 Property overview

Page	Properties	Access	Description
776	First	rw	First non-nil item.
777	FreeObjects	rw	Does the list own the objects or not?
776	Items	rw	Indexed access to objects in the list.
776	Last	rw	Last non-`Nil` object.
777	List	r	Internal list pointer.

51.11.4 TFPGObjectList.Create

Synopsis: Instantiate a new object list.

Declaration:
```
constructor Create(FreeObjects: Boolean)
```

Visibility: public

Description: `Create` instantiates a new object list. It will simply call the inherited constructor with the correct item size and will initialize TFPGObjectList.FreeObjects (777) with `FreeObjects`.

If `FreeObjects` is true, then the list owns the objects. Freeing or clearing the list will remove all objects from memory by calling the destructor. Deleting or removing an item from the list will also dispose of the element.

See also: TFPGObjectList.FreeObjects (777)

51.11.5 TFPGObjectList.Add

Synopsis: Add new object of class `T` to the list.

Declaration:
```
function Add(const Item: T) : Integer
```

Visibility: public

Description: `Add` adds a new item `Item` of class type `T` to the list and returns the position at which the item was added.

Errors: If the item could not be added, an EListError (747) exception is raised.

See also: TFPGObjectList.Extract (774), TFPGObjectList.Items (776), TFPGObjectList.IndexOf (774)

51.11.6 TFPGObjectList.Extract

Synopsis: Extract an item from the list.

Declaration: `function Extract(const Item: T) : T`

Visibility: public

Description: `Extract` removes `Item` from the list and returns the removed item, or `Nil` if it was not found.

The extracted object will not be destroyed even if the list owns the objects.

Errors: None.

See also: TFPSList.Delete (779)

51.11.7 TFPGObjectList.GetEnumerator

Synopsis: Return a list enumerator for `T`.

Declaration: `function GetEnumerator : TFPGListEnumeratorSpec`

Visibility: public

Description: `GetEnumerator` returns an enumerator for the elements in the list. It is a specialized version of TFPGListEnumerator (756).

See also: TFPGListEnumerator (756)

51.11.8 TFPGObjectList.IndexOf

Synopsis: Index of item.

Declaration: `function IndexOf(const Item: T) : Integer`

Visibility: public

Description: `IndexOf` returns the index of `Item` in the list, or -1 if the item does not appear in the list.

Errors: None.

See also: TFPGObjectList.Items (776), TFPGObjectList.Insert (775), TFPGObjectList.Add (773)

51.11.9 TFPGObjectList.Insert

Synopsis: Insert a new object in the list.

Declaration: `procedure Insert(Index: Integer; const Item: T)`

Visibility: public

Description: `Insert` inserts a new object (`Item`) in the list at position `Index`. The index is zero based and must be less than Count (784).

Errors: If an invalid index is specified, an EListError (747) exception is raised.

See also: TFPGList.Items (755), TFPGList.Insert (753), TFPSList.Add (779)

51.11.10 TFPGObjectList.AddList

Synopsis: Adds the elements from another list.

Declaration: `procedure AddList(Source: TFPGObjectList)`

Visibility: public

Description: `AddList` adds all the elements from list `Source` to the current list.

See also: TFPGList.AddList (754), TFPSList.AddList (782), TFPGObjectList.Add (773)

51.11.11 TFPGObjectList.Assign

Synopsis: Copy objects from Source list.

Declaration: `procedure Assign(Source: TFPGObjectList)`

Visibility: public

Description: `Assign` clears the list and copies all items in `Source` to the list. The source list must be of the same type as the destination list.

Take care if both the list owns the objects (i.e. have TFPGObjectList.FreeObjects (777) set to `True`), this may result to access violations.

See also: TFPSList.Clear (779), TFPGObjectList.Add (773)

51.11.12 TFPGObjectList.Remove

Synopsis: Remove an object from the list.

Declaration: `function Remove(const Item: T) : Integer`

Visibility: public

Description: `Remove` removes the object `Item` from the list, and returns the index of the removed item. If no item was removed, `-1` is returned. Only the first object is removed.

If the list owns the objects, (TFPGObjectList.FreeObjects (777) is set to `True`) then the object is freed.

Errors: None.

See also: TFPGObjectList.IndexOf (774), TFPSList.Delete (779), TFPGObjectList.FreeObjects (777)

51.11.13 TFPGObjectList.Sort

Synopsis: Sort the objects in the list.

Declaration: `procedure Sort(Compare: TCompareFunc)`

Visibility: public

Description: `Sort` sorts the elements in the list using the provided `Compare` function. The list passes 2 items to the compare function. The result of this function determines how the items will be sorted:

- If the result of this function is negative, the first object (`Item1`) is assumed to be 'less' than the second object (`Item2`) and will be moved before the second in the list.
- If the function result is positive, the first object (`Item1`) is assumed to be 'greater than' the second object (`Item2`)and will be moved after the second in the list.
- if the function result is zero, the objects are assumed to be 'equal' and no moving will take place.

Errors: None.

51.11.14 TFPGObjectList.First

Synopsis: First non-nil item.

Declaration: `Property First : T`

Visibility: public

Access: Read,Write

Description: `First` returns the first non-nil item. If no such element is present, `Nil` is returned.

See also: TFPSList.First (785), TFPGList.Last (755), TFPSList.Pack (783)

51.11.15 TFPGObjectList.Last

Synopsis: Last non-`Nil` object.

Declaration: `Property Last : T`

Visibility: public

Access: Read,Write

Description: `Last` returns the last non-`Nil` object. If no such element is present, `Nil`. is returned.

See also: TFPGObjectList.First (776), TFPSList.Last (785)

51.11.16 TFPGObjectList.Items

Synopsis: Indexed access to objects in the list.

Declaration: `Property Items[Index: Integer]: T; default`

Visibility: public

Access: Read,Write

Description: `Items` provides indexed access to the objects in the list. The objects can be get or set.

The index `Index` is zero based, and has a maximum value of Count-1 (784).

If the list owns the objects, (TFPGObjectList.FreeObjects (777) is set to `True`) then the previous object at position `Index` is freed when setting the property.

Errors: If an invalid index is used, an EListError (747) exception is raised.

See also: TFPSList.Count (784), TFPGObjectList.FreeObjects (777)

51.11.17 TFPGObjectList.List

Synopsis: Internal list pointer.

Declaration: `Property List : PTypeList`

Visibility: public

Access: Read

Description: `List` is the internal list of objects. It should not be used directly.

See also: TFPGObjectList.Items (776)

51.11.18 TFPGObjectList.FreeObjects

Synopsis: Does the list own the objects or not?

Declaration: `Property FreeObjects : Boolean`

Visibility: public

Access: Read,Write

Description: `FreeObjects` indicates whether the list owns the objects or not. If set to `True`, freeing or clearing the list will remove all objects from memory by calling the destructor. Deleting or removing an item from the list will also dispose of the element.

The initial value for this property is set in the constructor and is `True` by default.

See also: TFPGObjectList.FreeObjects (777)

51.12 TFPSList

51.12.1 Description

`TFPSList` can be seen as the generalized equivalent of the classes unit TFPList (746) list. It is used as a base class for the TFPGList (751), TFPGMap (757), TFPGObjectList (773), TFPGInterfacedObjectList (747) and TFPGMapInterfacedObjectData (762) generic classes.

This list is not meant to be used directly, it is an auxiliary class for the actual generic list classes.

See also: classes.TFPList (746), TFPGMap (757), TFPGObjectList (773), TFPGInterfacedObjectList (747), TFPGMapInterfacedObjectData (762)

51.12.2 Method overview

Page	Method	Description
779	Add	Add a new item to the list.
782	AddList	Adds the elements from another list.
782	Assign	Copy one list to another.
779	Clear	Clear the list.
778	Create	Create a new instance of `TFPSList`.
779	Delete	Delete an item from the list.
780	DeleteRange	Delete a range of elements.
778	Destroy	Destroy the list instance.
780	Error	Raise an `EListError` exception.
780	Exchange	Exchange two items in the list.
780	Expand	Expand the capacity of the list.
781	Extract	delete an element from the list.
781	IndexOf	Search an item in the list.
781	Insert	Insert a new item in the list.
779	ItemIsManaged	Indicates whether the list item is a managed type or not.
782	Move	Moves an item from one position in the list to another.
783	Pack	Remove empty items from the list.
782	Remove	Remove the item from the list.
783	Sort	Sort the list.

51.12.3 Property overview

Page	Properties	Access	Description
783	Capacity	rw	Current capacity of the list.
784	Count	rw	Current element count.
785	First	rw	Pointer to first non-empty item in the list.
784	Items	rw	Items in the list.
784	ItemSize	r	Size of the items in the list.
785	Last	rw	Pointer to last non-empty item in the list.
784	List	r	Internal list pointer.

51.12.4 TFPSList.Create

Synopsis: Create a new instance of `TFPSList`.

Declaration:
```
constructor Create(AItemSize: Integer)
```

Visibility: public

Description: `Create` creates a new instance of `TFPSList`, and initializes the item size to `ItemSize`, which defaults to the size of a pointer.

See also: TFPSList.ItemSize (784), TFPSList.Destroy (778)

51.12.5 TFPSList.Destroy

Synopsis: Destroy the list instance.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` clears and cleans up the list instance. Depending on the descendant, this may also remove the items in the list from memory.

See also: TFPSList.Clear (779)

51.12.6 TFPSList.ItemIsManaged

Synopsis: Indicates whether the list item is a managed type or not.

Declaration: `class function ItemIsManaged : Boolean; Virtual`

Visibility: public

Description: `ItemIsManaged` indicates whether the list item type is a managed type or not. If not, faster codepaths are sometimes taken in the code.

51.12.7 TFPSList.Add

Synopsis: Add a new item to the list.

Declaration: `function Add(Item: Pointer) : Integer`

Visibility: public

Description: `Add` adds the item pointed to by `Item` to the list. It is not the pointer `Item` itself which is added to the list, but rather the TFPSList.ItemSize (784) bytes of memory to which `Item` is pointing.

The function returns the index of the newly added item.

Errors: If the maximum list size is reached, an EListError (747) is raised.

See also: TFPSList.Delete (779), TFPSList.Items (784), TFPSList.Clear (779)

51.12.8 TFPSList.Clear

Synopsis: Clear the list.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` removes all the items in the list. Depending on the descendent, the list items themselves may be cleared as well.

See also: TFPSList.Delete (779), TFPSList.Items (784), TFPSList.Add (779)

51.12.9 TFPSList.Delete

Synopsis: Delete an item from the list.

Declaration: `procedure Delete(Index: Integer)`

Visibility: public

Description: `Delete` deletes the item at position `Index` from the list. Depending on the descendent, the list items itself may be cleared as well.

Errors: If `Index` is out of bounds, an EListError (747) exception is raised.

See also: TFPSList.Clear (779), TFPSList.Items (784), TFPSList.Add (779)

51.12.10 TFPSList.DeleteRange

Synopsis: Delete a range of elements.

Declaration: `procedure DeleteRange(IndexFrom: Integer; IndexTo: Integer)`

Visibility: public

Description: `DeleteRange` deletes elements from `IndexFrom` till `IndexTo`. Both indexes are zero based, and `IndexTo` must be bigger than `IndexFrom`. Using this method results in less moving of data in memory, and as such is more effective than deleting the elements one by one using TFPSList.Delete (779).

Errors: If invalid indexes are specified, an EListError (747) exception will be raised.

See also: EListError (747), TFPSList.Delete (779)

51.12.11 TFPSList.Error

Synopsis: Raise an `EListError` exception.

Declaration: `class procedure Error(const Msg: string; Data: PtrInt)`

Visibility: public

Description: `Error` is an auxiliary routine which raises an EListError (747) exception formatted from `Msg` and `Data`.

See also: EListError (747)

51.12.12 TFPSList.Exchange

Synopsis: Exchange two items in the list.

Declaration: `procedure Exchange(Index1: Integer; Index2: Integer)`

Visibility: public

Description: `Exchange` will exchange 2 items at positions `Index1` and `Index2` in the list.

Errors: If `Index1` or `Index2` are out of bounds, an EListError (747) exception is raised.

51.12.13 TFPSList.Expand

Synopsis: Expand the capacity of the list.

Declaration: `function Expand : TFPSList`

Visibility: public

Description: `Expand` will expand the capacity of the list, and returns itself.

Errors: If the size will become larger than MaxListSize (746) an EListError (747) exception will be raised.

See also: MaxListSize (746), TFPSList.Capacity (783)

51.12.14 TFPSList.Extract

Synopsis: delete an element from the list.

Declaration:
```
procedure Extract(Item: Pointer; ResultPtr: Pointer)
```

Visibility: public

Description: `Extract` removes the item pointed to by `Item` from the list. It returns a pointer to the actually removed item from the list. The item is searched using TFPSList.IndexOf (781). If the item is not found, nil is returned.

Some descendents own the items in the list. `Extract` will not dispose of the item, as TFPSList.Delete (779) does.

Errors: None.

See also: TFPSList.Delete (779), TFPSList.Add (779), TFPSList.IndexOf (781)

51.12.15 TFPSList.IndexOf

Synopsis: Search an item in the list.

Declaration:
```
function IndexOf(Item: Pointer) : Integer
```

Visibility: public

Description: `IndexOf` searches in the list for the item pointed to by `Item`, and returns the position (0-based index) of the item in the list, or -1 if it was not found. The items are compared using sysutils.CompareMem (746), so an exact memory layout match.

See also: TFPSList.Extract (781)

51.12.16 TFPSList.Insert

Synopsis: Insert a new item in the list.

Declaration:
```
procedure Insert(Index: Integer; Item: Pointer)
function Insert(Index: Integer) : Pointer
```

Visibility: public

Description: `Insert` comes in 2 overloaded version. The version without `Item` creates a slot for a new item at position `Index` in the list, and returns a pointer to the new slot. The slot will be of size TFPSList.ItemSize (784).

The version with `Item` argument will allocate a slot in the list at position `Index` and will copy the item pointed to by `Item` to the slot in the list.

In both cases, `Index` must be 0-based.

Errors: If `Index` is invalid, an EListError (747) exception will be raised.

See also: TFPSList.Add (779), TFPSList.Delete (779), TFPSList.Extract (781)

51.12.17 TFPSList.Move

Synopsis: Moves an item from one position in the list to another.

Declaration: `procedure Move(CurIndex: Integer; NewIndex: Integer)`

Visibility: public

Description: `Move` moves the item at position `CurIndex` to position `NewIndex`. This is done by storing the value at position `CurIndex`, deleting the pointer at position `CurIndex`, and reinserting the value at position `NewIndex`

Errors: If `CurIndex` or `Newindex` are not inside the valid range of indices, an EListError (747) exception is raised.

See also: Exchange (780)

51.12.18 TFPSList.Assign

Synopsis: Copy one list to another.

Declaration: `procedure Assign(Obj: TFPSList)`

Visibility: public

Description: `Assign` clears the list and will add all items from `Obj` to the list. The items are copied one by one.

Errors: If the TFPSList.ItemSize (784) differs for the two lists, an EListError (747) exception is raised.

See also: TFPSList.Add (779)

51.12.19 TFPSList.AddList

Synopsis: Adds the elements from another list.

Declaration: `procedure AddList(Obj: TFPSList)`

Visibility: public

Description: `AddList` adds all the elements from list `Obj` to the current list. A check is done that the 2 lists have the same element size.

Errors: If the lists have different element size, an EListError (747) exception will be raised.

See also: TFPSList.Add (779), EListError (747)

51.12.20 TFPSList.Remove

Synopsis: Remove the item from the list.

Declaration: `function Remove(Item: Pointer) : Integer`

Visibility: public

Description: `Remove` searches `Item` in the list, and deletes it from the list. It returns the index of the item that was removed, or -1 if it was not found. Only the first match is removed.

If a descendent of `TFPSList` owns the object of the list, the item is removed from memory. If this is not desired, then TFPSList.Extract (781) must be used instead.

See also: TFPSList.Extract (781), TFPSList.IndexOf (781)

51.12.21 TFPSList.Pack

Synopsis: Remove empty items from the list.

Declaration: `procedure Pack`

Visibility: public

Description: `Pack` will remove all empty items from the list. An item is considered to be empty if the memory location where the item is stored contains only zero bytes.

See also: TFPSList.Clear (779), TFPSList.Sort (783)

51.12.22 TFPSList.Sort

Synopsis: Sort the list.

Declaration: `procedure Sort(Compare: TFPSListCompareFunc)`

Visibility: public

Description: `Sort>` sorts the items in the list. Two pointers are compared by passing them to the `Compare` function. The result of this function determines how the pointers will be sorted:

- If the result of this function is negative, the first item is assumed to be 'less' than the second and will be moved before the second item in the list.
- If the function result is positive, the first item is assumed to be 'greater than' the second and will be moved after the second item in the list.
- if the function result is zero, the pointers are assumed to be 'equal' and no moving will take place.

The sort is done using a quicksort algorithm.

See also: TFPSListCompareFunc (746)

51.12.23 TFPSList.Capacity

Synopsis: Current capacity of the list.

Declaration: `Property Capacity : Integer`

Visibility: public

Access: Read,Write

Description: `Capacity` is the current capacity (maximum amount of elements) of the list. The list capacity will expand automatically if an item is added and the capacity is reached (i.e. TFPSList.Count (784) equals `capacity`. Expanding the list is an expensive operation involving reallocation of memory and moving of list data in memory, so capacity can be set to a large amount to avoid frequent reallocations.

See also: TFPSList.Count (784), TFPSList.Expand (780), TFPSList.Items (784)

51.12.24 TFPSList.Count

Synopsis: Current element count.

Declaration: `Property Count : Integer`

Visibility: public

Access: Read,Write

Description: `Count` is the current amount of elements in the list. It is initially zero. A valid item index is always a value between zero and `Count-1`.

See also: TFPSList.Items (784), TFPSList.Capacity (783)

51.12.25 TFPSList.Items

Synopsis: Items in the list.

Declaration: `Property Items[Index: Integer]: Pointer; default`

Visibility: public

Access: Read,Write

Description: `Items` provides indexed access to the items in the list, the returned pointers are not the actual list items, but pointers to the elements in the list. The items can be get or set.

When assigning to the `Items` property, the memory area pointed to by the assigned pointer is copied to the list. Exactly TFPSList.ItemSize (784) bytes are copied.

The index `Index` is zero based, and has a maximum value of Count-1 (784).

Errors: If an invalid index is used, an EListError (747) exception is raised.

See also: TFPSList.ItemSize (784), TFPSList.Count (784)

51.12.26 TFPSList.ItemSize

Synopsis: Size of the items in the list.

Declaration: `Property ItemSize : Integer`

Visibility: public

Access: Read

Description: `ItemSize` is the memory size of the items in the list. It is specified in the constructor and cannot be changed during the lifetime of the list.

See also: TFPSList.Create (778)

51.12.27 TFPSList.List

Synopsis: Internal list pointer.

Declaration: `Property List : PByte`

Visibility: public

Access: Read

Description: `List` is a pointer to the memory area used for the items in the list. It should not be manipulated.

51.12.28 TFPSList.First

Synopsis: Pointer to first non-empty item in the list.

Declaration: `Property First : Pointer`

Visibility: public

Access: Read,Write

Description: `First` returns the value of the first non-empty item in the list. An item is considered empty if consists of TFPSList.ItemSize (784) zero bytes.

If there are no non-empty items in the list, then `Nil` is returned.

See also: TFPSList.Last (785), TFPSList.Pack (783)

51.12.29 TFPSList.Last

Synopsis: Pointer to last non-empty item in the list.

Declaration: `Property Last : Pointer`

Visibility: public

Access: Read,Write

Description: `Last` returns the value of the last non-empty item in the list. An item is considered empty if consists of TFPSList.ItemSize (784) zero bytes.

If there are no non-empty items in the list, then `Nil` is returned.

See also: TFPSList.Last (785), TFPSList.Pack (783)

51.13 TFPSMap

51.13.1 Description

`TFPSMap` can be used to create a map for any type `T` that does not require reference counting (such as interfaced objects). It will specialize to a map which is a generalized list with an arbitrary type as the list index (called the key).

This class should normally not be used directly, instead use one of the generic map objects such as TFPGMap (757).

See also: TFPGMap (757)

51.13.2 Method overview

Page	Method	Description
786	Add	Add a key, value pair to the map.
786	Create	Create a new map with given key and data size.
786	Find	Find data using the associated key.
787	IndexOf	Index of key pointed to by `AKey`.
787	IndexOfData	Index of data item `AData`.
787	Insert	Insert a new slot for key and associated data item in the list.
788	InsertKey	Insert a key in the list.
788	InsertKeyData	Insert a key and associated in the list.
788	Remove	Remove a key/value pair from the map.
788	Sort	Sort the list according to key.

51.13.3 Property overview

Page	Properties	Access	Description
790	Data	rw	Indexed access to the locations of all data items.
789	DataSize	r	Size (in bytes) for the data associated with keys.
789	Duplicates	rw	What to do with duplicate key values.
790	KeyData	rw	Access to data locations using key.
790	Keys	rw	Indexed access to the locations of all keys.
789	KeySize	r	Size (in bytes) for the key.
791	OnDataPtrCompare	rw	Callback to compare 2 data items.
791	OnKeyPtrCompare	rw	Callback to compare 2 keys.
791	OnPtrCompare	rw	Alias for `OnKeyPtrCompare`.
790	Sorted	rw	Is the map permanently sorted on key ?

51.13.4 TFPSMap.Create

Synopsis: Create a new map with given key and data size.

Declaration:
```
constructor Create(AKeySize: Integer; ADataSize: Integer)
```

Visibility: public

Description: `Create` instantiates a new `TFPSMap` instance and initializes TFPSMap.KeySize (789) and TFPSMap.DataSize (789) with `AKeySize` and `ADataSize`, respectively. It also initializes the TFPSMap.OnDataPtrCompare (791) and TFPSMap.OnKeyPtrCompare (791) properties to functions that compare memory blocks.

Errors: None.

See also: TFPSMap.Destroy (785), TFPSMap.KeySize (789), TFPSMap.DataSize (789), TFPSMap.OnDataPtrCompare (791), TFPSMap.OnKeyPtrCompare (791)

51.13.5 TFPSMap.Add

Synopsis: Add a key, value pair to the map.

Declaration:
```
function Add(AKey: Pointer; AData: Pointer) : Integer
function Add(AKey: Pointer) : Integer
```

Visibility: public

Description: `Add` adds the memory pointed to by `AData` to the map using the memory pointed to by `AKey` as the key. If no data is specified, it allocates a slot for `AKey` and returns a pointer to that slot.

Errors: If the maximum amount of values is reached, `Add` will raise an EListError (747) exception.

See also: TFPSMap.Insert (787), TFPSMap.IndexOf (787)

51.13.6 TFPSMap.Find

Synopsis: Find data using the associated key.

Declaration:
```
function Find(AKey: Pointer; out Index: Integer) : Boolean
```

Visibility: public

Description: `Find` searches for the first key less than or equal to `AKey` and returns its index in the list in `Index`. It returns `True` if an exact match for the key was found. It returns `False` if the key was not found, and `Index` is then set to `-1`

This function performs a binary search using the key comparing function specified in OnKeyPtrCompare (791).

Errors: if `OnKeyPtrCompare` is not set, an access violation will occur.

See also: TFPSMap.OnKeyPtrCompare (791)

51.13.7 TFPSMap.IndexOf

Synopsis: Index of key pointed to by `AKey`.

Declaration:
```
function IndexOf(AKey: Pointer) : Integer
```

Visibility: public

Description: `IndexOf` returns the index of the element with a key pointed to by `AKey`. It returns -1 if the key was not found.

If the list is sorted, then a binary search is performed, otherwise a linear search is used to find the key.

Errors: None.

See also: TFPSMap.Find (786), TFPSMap.IndexOfData (787), TFPSMap.Keys (790), TFPSMap.Data (790)

51.13.8 TFPSMap.IndexOfData

Synopsis: Index of data item `AData`.

Declaration:
```
function IndexOfData(AData: Pointer) : Integer
```

Visibility: public

Description: `IndexOfData` returns the index of the element with data pointed to by `AData`. It returns -1 if the data item was not found. The search is always performed using a linear search.

See also: TFPSMap.Find (786), TFPSMap.IndexOf (787), TFPSMap.Keys (790), TFPSMap.Data (790)

51.13.9 TFPSMap.Insert

Synopsis: Insert a new slot for key and associated data item in the list.

Declaration:
```
function Insert(Index: Integer) : Pointer
procedure Insert(Index: Integer; out AKey: Pointer; out AData: Pointer)
```

Visibility: public

Description: `Insert` will allocate a new slot in the list. It returns a pointer to the new slot. If `Akey` and `AData` are given, then they will point to the positions in the slot for the key and data items.

Errors: If the maximum amount of values is reached or an invalid index is specified, `Insert` will raise an EListError (747) exception.

See also: TFPSMap.Add (786), TFPSMap.InsertKey (788), TFPSMap.InsertKeyData (788)

51.13.10 TFPSMap.InsertKey

Synopsis: Insert a key in the list.

Declaration: `procedure InsertKey(Index: Integer; AKey: Pointer)`

Visibility: public

Description: `InsertKey` will allocate a new slot in the list for a key value as pointed to by `AKey`, and copy the key pointed to by `AKey` to the slot.

Errors: If the maximum amount of values is reached or an invalid index is specified, `InsertKey` will raise an EListError (747) exception.

See also: TFPSMap.Add (786), TFPSMap.Insert (787), TFPSMap.InsertKeyData (788)

51.13.11 TFPSMap.InsertKeyData

Synopsis: Insert a key and associated in the list.

Declaration: `procedure InsertKeyData(Index: Integer; AKey: Pointer; AData: Pointer)`

Visibility: public

Description: `InsertKeyData` will allocate a new slot in the list for a key value as pointed to by `AKey`, and copy the key pointed to by `AKey` as well as the data pointed to by `AData` to the newly allocated slot.

Errors: If the maximum amount of values is reached or an invalid index is specified, `InsertKeyData` will raise an EListError (747) exception.

See also: TFPSMap.Add (786), TFPSMap.Insert (787), TFPSMap.InsertKey (788)

51.13.12 TFPSMap.Remove

Synopsis: Remove a key/value pair from the map.

Declaration: `function Remove(AKey: Pointer) : Integer`

Visibility: public

Description: `Remove` removes the key/value pair pointing to by `AKey` from the map. It returns the index of `Akey` prior to removal from the list. If `AKey` was not found, -1 is returned.

Errors: None.

See also: TFPSList.Delete (779), TFPSMap.IndexOf (787)

51.13.13 TFPSMap.Sort

Synopsis: Sort the list according to key.

Declaration: `procedure Sort`

Visibility: public

Description: `Sort` sorts the list according to the key value, using the compare function provided in TFPSMap.OnKeyPtrCompare (791)

Errors: None.

See also: TFPSMap.OnKeyPtrCompare (791), TFPSMap.OnDataPtrCompare (791)

51.13.14 TFPSMap.Duplicates

Synopsis: What to do with duplicate key values.

Declaration: `Property Duplicates : TDuplicates`

Visibility: public

Access: Read,Write

Description: `Duplicates` can be set to determine what to do with duplicate key values in the map:

dupIgnoreIgnore the new item, do not add it to the list.

dupAcceptAccept duplicates, adding them to the list.

dupErrorRaise an error when an attempt is made to add a duplicate.

The value is ignored if Sorted (790) is `False`.

See also: TFPSMap.Sorted (790), types.TDuplicates (746)

51.13.15 TFPSMap.KeySize

Synopsis: Size (in bytes) for the key.

Declaration: `Property KeySize : Integer`

Visibility: public

Access: Read

Description: `KeySize` is the size (in bytes) for the keys in the map. This size is initialized during list construction and defaults to the size of a pointer.

See also: TFPSMap.Create (786), TFPSMap.DataSize (789), TFPSMap.Keys (790), TFPSMap.KeyData (790)

51.13.16 TFPSMap.DataSize

Synopsis: Size (in bytes) for the data associated with keys.

Declaration: `Property DataSize : Integer`

Visibility: public

Access: Read

Description: `DataSize` is the size (in bytes) for the data in the map. This size is initialized during list construction and defaults to the size of a pointer.

See also: TFPSMap.Create (786), TFPSMap.KeySize (789), TFPSMap.Keys (790), TFPSMap.KeyData (790), TFPSMap.Data (790)

51.13.17 TFPSMap.Keys

Synopsis: Indexed access to the locations of all keys.

Declaration: `Property Keys[Index: Integer]: Pointer`

Visibility: public

Access: Read,Write

Description: `Keys` provides indexed access to all keys. It does not return the key, but returns a pointer to the key value. When setting the `Keys` property, a pointer to the key value must be set, and the key value is copied from the pointer.

See also: TFPSMap.KeySize (789), TFPSMap.KeyData (790), TFPSMap.Data (790)

51.13.18 TFPSMap.Data

Synopsis: Indexed access to the locations of all data items.

Declaration: `Property Data[Index: Integer]: Pointer`

Visibility: public

Access: Read,Write

Description: `Data` provides indexed access to all data. It does not return the actual data, but returns a pointer to the data. When setting the `Data` property, a pointer to the data value must be set, and the data value is copied from the pointer.

See also: TFPSMap.DataSize (789), TFPSMap.KeyData (790), TFPSMap.Keys (790)

51.13.19 TFPSMap.KeyData

Synopsis: Access to data locations using key.

Declaration: `Property KeyData[Key: Pointer]: Pointer; default`

Visibility: public

Access: Read,Write

Description: `KeyData` provides access to the data items, using their key value (as pointed to by `AKey`) as an index. When reading a non-existent key value, `Nil` is returned. If the key is found, a pointer to the associated data's location is returned. When writing, the key pointed to by `Key` is added if it was not present, and the data data is copied from the written pointer.

See also: TFPSMap.DataSize (789), TFPSMap.KeyData (790), TFPSMap.Data (790), TFPSMap.Keys (790), TFPSMap.KeySize (789)

51.13.20 TFPSMap.Sorted

Synopsis: Is the map permanently sorted on key ?

Declaration: `Property Sorted : Boolean`

Visibility: public

Access: Read,Write

Description: `Sorted` can be set to true to keep the map permanently sorted on key value. The sorting happens using TFPSMap.OnKeyPtrCompare (791).

See also: TFPSMap.OnKeyPtrCompare (791), TFPSMap.Sort (788), TFPSMap.Duplicates (789)

51.13.21 TFPSMap.OnPtrCompare

Synopsis: Alias for `OnKeyPtrCompare`.

Declaration: `Property OnPtrCompare : TFPSListCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnPtrCompare` is a deprecated alias for OnKeyPtrCompare (791).

See also: TFPSMap.OnKeyPtrCompare (791), TFPSMap.OnDataPtrCompare (791)

51.13.22 TFPSMap.OnKeyPtrCompare

Synopsis: Callback to compare 2 keys.

Declaration: `Property OnKeyPtrCompare : TFPSListCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnKeyPtrCompare` is used to compare the values of 2 keys. By default it simply compares the byte values of the key memory block. It can be set to any function that performs another comparison. (e.g. a function that treats the memory blocks as a string pointer and compare the actual strings).

This function is used to sort the list or find a key.

See also: TFPSListCompareFunc (746), TFPSMap.OnDataPtrCompare (791), TFPSMap.Sort (788), TFPSMap.Find (786), TFPSMap.IndexOf (787)

51.13.23 TFPSMap.OnDataPtrCompare

Synopsis: Callback to compare 2 data items.

Declaration: `Property OnDataPtrCompare : TFPSListCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnKeyPtrCompare` is used to compare the values of 2 keys. By default it simply compares the byte values of the key memory block. It can be set to any function that performs another comparison. (e.g. a function that treats the memory blocks as a string pointer and compare the actual strings).

This function is used to find a data item (IndexOf (787)).

See also: TFPSListCompareFunc (746), TFPSMap.OnKeyPtrCompare (791), TFPSMap.IndexOfData (787)

Chapter 52

Reference for unit 'fpwidestring'

52.1 Used units

Table 52.1: Used units by unit 'fpwidestring'

Name	Page
System	1362
unicodedata	2098

52.2 Overview

`fpwidestring` implements Unicode string support for the Free Pascal RTL using native Object Pascal routines. It is meant to be used on operating systems where the operating system does not natively support Unicode transformations and operations.

In general, it is sufficient to include the unit in the uses clause of a program. The initialization code of the unit will set the Unicode string manager of the system unit to the object pascal implementation contained in this unit.

This unit needs Unicode collation and character set tables in order to be able to do its work correctly. These must be registered using the routines of the unicodedata (792) unit: the FPC project distributes some Unicode collation data in `.bco` files which can be loaded using the LoadCollation (2109) routine.

In order for sorting and comparing of strings to work, a collation must be used. The collation in general depends on the internationalization of the application. Since the system unit does not know about collations, the collation must be set in the `fpWideString` unit using the SetActiveCollation (793) function. The collation can be set on a per-thread basis.

New threads get DefaultCollationName (793) as the active collation name.

The fpwidestring unit performs conversions between Unicode and single-byte ansistring conversions (excluding UTF8). Support for various single-byte encodings are based on the charset (792) unit. This unit can be used to load single byte code pages. Various code page units such as cp895, cp932, cp950 are provided by the "rtl-unicode" package.

The fpwidestring requires at least the Default Unicode Collation Element Table to be registered (called `DUCET`). The DUCET encoding is provided by the unicodeducet unit. More information can be found in the unicodedata (792) unit.

52.3 Constants, types and variables

52.3.1 Variables

```
DefaultCollationName : UnicodeString = ''
```

`DefaultCollationName` is the collation name for new threads. It is empty by default.

52.4 Procedures and functions

52.4.1 GetActiveCollation

Synopsis: Return the currently active collation for the current thread.

Declaration:
```
function GetActiveCollation : PUCA_DataBook
```

Visibility: default

Description: `GetActiveCollation` returns the currently active collation, for the current thread, as set using the SetActiveCollation (793) function.

New threads get DefaultCollationName (793) as the active collation name. The collation can be changed per thread using the SetActiveCollation (793) function.

Errors: None.

See also: SetActiveCollation (793), DefaultCollationName (793)

52.4.2 SetActiveCollation

Synopsis: Set the active collation for the current thread.

Declaration:
```
function SetActiveCollation(const AName: UnicodeString) : Boolean
function SetActiveCollation(const ACollation: PUCA_DataBook) : Boolean
```

Visibility: default

Description: `SetActiveCollation` sets the collation used in the current thread of the application. This can be done using the name of the collation (`AName`) as registered or in the unicodedata (792) unit or using the actual collation data (`ACollation`).

Errors: If the collation `AName` was not found, or `ACollation` is nil, then `False` is returned.

See also: GetActiveCollation (793), DefaultCollationName (793)

Chapter 53

Reference for unit 'getopts'

53.1 Used units

Table 53.1: Used units by unit 'getopts'

Name	Page
System	1362

53.2 Overview

This document describes the GETOPTS unit for Free Pascal. It was written for Linux by Michael Van Canneyt. It now also works for all supported platforms.

The getopts unit provides a mechanism to handle command-line options in a structured way, much like the GNU getopts mechanism. It allows you to define the valid options for you program, and the unit will then parse the command-line options for you, and inform you of any errors.

53.3 Constants, types and variables

53.3.1 Constants

```
EndOfOptions = #255
```

Returned by getopt (796),getlongopts (795) to indicate that there are no more options.

```
No_Argument = 0
```

Specifies that a long option does not take an argument.

```
Optional_Argument = 2
```

Specifies that a long option optionally takes an argument.

```
OptSpecifier : Set of Char = ['-']
```

Character indicating an option on the command-line.

```
Required_Argument = 1
```

Specifies that a long option needs an argument.

53.3.2 Types

```
Orderings = (require_order,permute,return_in_order)
```

Table 53.2: Enumeration values for type Orderings

Value	Explanation
permute	Change command-line options.
require_order	Don't touch the ordering of the command-line options.
return_in_order	Return options in the correct order.

Command-line ordering options.

```
POption = ^TOption
```

Pointer to TOption (799) record.

53.3.3 Variables

```
OptArg : string
```

Set to the argument of an option, if the option needs one.

```
OptErr : Boolean
```

Indicates whether `getopt()` prints error messages.

```
OptInd : LongInt
```

when all options have been processed, `optind` is the index of the first non-option parameter. This is a read-only variable. Note that it can become equal to `paramcount+1`.

```
OptOpt : Char
```

In case of an error, contains the character causing the error.

53.4 Procedures and functions

53.4.1 GetLongOpts

Synopsis: Return next long option.

Declaration: `function GetLongOpts(ShortOpts: string; LongOpts: POption; var Longind: LongInt) : Char`

Visibility: default

Description: Returns the next option found on the command-line, taking into account long options as well. If no more options are found, returns `EndOfOptions`. If the option requires an argument, it is returned in the `OptArg` variable.

if any of the option definitions in `LongOpts` array has the `Flag` pointer set, then the return value is the null character (`#0` or `char(0)`) and the actual option letter is written in the location pointed to by `Flag`.

`ShortOptions` is a string containing all possible one-letter options. (see Getopt (796) for its description and use) `LongOpts` is a pointer to the first element of an array of `Option` records, the last of which needs a name of zero length.

The function tries to match the names even partially (i.e. `-app` will match e.g. the `append` option), but will report an error in case of ambiguity.

If the option needs an argument, set `Has_arg` to `Required_argument` (1), if the option optionally has an argument, set `Has_arg` to `Optional_argument` (2). If the option needs no argument, set `Has_arg` to zero.

Required arguments can be specified in two ways :

1. Pasted to the option : `-option=value`
2. As a separate argument : `-option value`

Optional arguments can only be specified through the first method.

Errors: see Getopt (796).

See also: Getopt (796)

53.4.2 GetOpt

Synopsis: Get next short option.

Declaration: `function GetOpt(ShortOpts: string) : Char`

Visibility: default

Description: Returns the next option found on the command-line. If no more options are found, returns `EndOfOptions`. If the option requires an argument, it is returned in the `OptArg` variable.

`ShortOptions` is a string containing all possible one-letter options. If a letter is followed by a colon (:), then that option needs an argument. If a letter is followed by 2 colons, the option has an optional argument. If the first character of `shortoptions` is a `'+'` then options following a non-option are regarded as non-options (standard Unix behavior). If it is a `'-'`, then all non-options are treated as arguments of a option with character `#0`. This is useful for applications that require their options in the exact order as they appear on the command-line. If the first character of `shortoptions` is none of the above, options and non-options are permuted, so all non-options are behind all options. This allows options and non-options to be in random order on the command line.

Errors: Errors are reported through giving back a `'?'` character. `OptOpt` then gives the character which caused the error. If `OptErr` is `True` then getopt prints an error-message to `stdout`.

See also: GetLongOpts (795)

Listing: ./examples/optex/optex.pp

```
program testopt;

{ Program to depmonstrate the getopts function. }

{
  Valid calls to this program are
  optex --verbose --add me --delete you
  optex --append --create child
  optex -ab -c me -d you
  and so on
}
uses getopts;

var c : char;
    optionindex : Longint;
    theopts : array[1..7] of TOption;

begin
  with theopts[1] do
   begin
    name:='add';
    has_arg:=1;
    flag:=nil;
    value:=#0;
  end;
  with theopts[2] do
   begin
    name:='append';
    has_arg:=0;
    flag:=nil;
    value:=#0;
  end;
  with theopts[3] do
   begin
    name:='delete';
    has_arg:=1;
    flag:=nil;
    value:=#0;
  end;
  with theopts[4] do
   begin
    name:='verbose';
    has_arg:=0;
    flag:=nil;
    value:=#0;
  end;
  with theopts[5] do
   begin
    name:='create';
    has_arg:=1;
    flag:=nil;
    value:='c'
  end;
  with theopts[6] do
   begin
    name:='file';
    has_arg:=1;
    flag:=nil;
```

```
    value:=#0;
  end;
  with theopts[7] do
   begin
    name:='';
    has_arg:=0;
    flag:=nil;
  end;
  c:=#0;
  repeat
    c:=getlongopts('abc:d:012',@theopts[1],optionindex);
    case c of
      '1','2','3','4','5','6','7','8','9' :
        begin
        writeln ('Got optind : ',c)
        end;
      #0 : begin
           write ('Long option : ',theopts[optionindex].name);
           if theopts[optionindex].has_arg>0 then
             writeln (' With value  : ',optarg)
           else
             writeln
           end;
      'a' : writeln ('Option a.');
      'b' : writeln ('Option b.');
      'c' : writeln ('Option c : ', optarg);
      'd' : writeln ('Option d : ', optarg);
      '?',':' : writeln ('Error with opt : ',optopt);
   end; { case }
 until c=endofoptions;
 if optind<=paramcount then
    begin
    write ('Non options : ');
    while optind<=paramcount do
      begin
      write (paramstr(optind),' ');
      inc(optind)
      end;
    writeln
    end
end.
```

53.5 TOption

```
TOption = record
public
  Name : string;
  Has_arg : Integer;
  Flag
  : PChar;
  Value : Char;
  procedure SetOption(const aName: string
  ; AHas_Arg: Integer;
                    AFlag: PChar; AValue: Char
  );
```

```
end
```

The TOption type is used to communicate the long options to GetLongOpts (795). The Name field is the name of the option. Has_arg specifies if the option wants an argument, Flag is a pointer to a char, which is set to Value, if it is non-nil.

53.5.1 Method overview

Page	Method	Description
799	SetOption	Set all option fields in 1 call.

53.5.2 TOption.SetOption

Synopsis: Set all option fields in 1 call.

Declaration:
```
procedure SetOption(const aName: string; AHas_Arg: Integer;
                    AFlag: PChar; AValue: Char)
```

Visibility: public

Description: SetOption can be used to set all fields in 1 single call:

- Name is set to aName
- Has_arg is set to aHas_arg
- Flag is set to aFlag
- Value is set to aValue

Chapter 54

Reference for unit 'go32'

54.1 Used units

Table 54.1: Used units by unit 'go32'

Name	Page
System	1362

54.2 Overview

This document describes the GO32 unit for the Free Pascal compiler under dos. It was donated by Thomas Schatzl (tom_at_work@geocities.com), for which my thanks. This unit was first written for dos by Florian Klaempfl.

Only the GO32V2 DPMI mode is discussed by me here due to the fact that new applications shouldn't be created with the older GO32V1 model. The go32v2 version is much more advanced and better. Additionally a lot of functions only work in DPMI mode anyway. I hope the following explanations and introductions aren't too confusing at all. If you notice an error or bug send it to the FPC mailing list or directly to me. So let's get started and happy and error free coding I wish you.... Thomas Schatzl, 25. August 1998

54.3 Real mode callbacks.

The callback mechanism can be thought of as the converse of calling a real mode procedure (i.e. interrupt), which allows your program to pass information to a real mode program, or obtain services from it in a manner that's transparent to the real mode program. In order to make a real mode callback available, you must first get the real mode callback address of your procedure and the selector and offset of a register data structure. This real mode callback address (this is a segment:offset address) can be passed to a real mode program via a software interrupt, a dos memory block or any other convenient mechanism. When the real mode program calls the callback (via a far call), the DPMI host saves the registers contents in the supplied register data structure, switches into protected mode, and enters the callback routine with the following settings:

- interrupts disabled

- %CS:%EIP = 48 bit pointer specified in the original call to get_rm_callback (826)
- %DS:%ESI = 48 bit pointer to real mode SS:SP
- %ES:%EDI = 48 bit pointer of real mode register data structure.
- %SS:%ESP = locked protected mode stack
- All other registers undefined

The callback procedure can then extract its parameters from the real mode register data structure and/or copy parameters from the real mode stack to the protected mode stack. Recall that the segment register fields of the real mode register data structure contain segment or paragraph addresses that are not valid in protected mode. Far pointers passed in the real mode register data structure must be translated to virtual addresses before they can be used with a protected mode program. The callback procedure exits by executing an IRET with the address of the real mode register data structure in %ES:%EDI, passing information back to the real mode caller by modifying the contents of the real mode register data structure and/or manipulating the contents of the real mode stack. The callback procedure is responsible for setting the proper address for resumption of real mode execution into the real mode register data structure; typically, this is accomplished by extracting the return address from the real mode stack and placing it into the %CS:%EIP fields of the real mode register data structure. After the IRET, the DPMI host switches the CPU back into real mode, loads ALL registers with the contents of the real mode register data structure, and finally returns control to the real mode program. All variables and code touched by the callback procedure MUST be locked to prevent page faults.

See also: get_rm_callback (826), free_rm_callback (821), lock_code (837), lock_data (837)

54.4 Executing software interrupts.

Simply execute a realintr() call with the desired interrupt number and the supplied register data structure. But some of these interrupts require you to supply them a pointer to a buffer where they can store data to or obtain data from in memory. These interrupts are real mode functions and so they only can access the first Mb of linear address space, not FPC's data segment. For this reason FPC supplies a pre-initialized dos memory location within the GO32 unit. This buffer is internally used for dos functions too and so it's contents may change when calling other procedures. It's size can be obtained with tb_size (847) and it's linear address via transfer_buffer (848). Another way is to allocate a completely new dos memory area via the global_dos_alloc (833) function for your use and supply its real mode address.

See also: tb_size (847), transfer_buffer (848), global_dos_alloc (833), global_dos_free (835), realintr (839)

Listing: ./examples/go32ex/softint.pp

```
{ Executes a real mode software interrupt

Exactly the interrupt call to get the DOS version.

get DOS version Int 21h / function 30h
Input:
        AH = $30
        AL = $1
Return:
        AL = major version number
        AH = minor version number
}
```

```
uses
        go32;

var
        r : trealregs;

begin
        r.ah := $30;
        r.al := $01;
        realintr($21, r);
        Writeln('DOS v', r.al,'.',r.ah, ' detected');
end.
```

Listing: ./examples/go32ex/rmpmint.pp

```
{ This example shows the difference between protected and real mode
interrupts; it redirects the protected mode handler to an own handler
which returns an impossible function result and calls it afterwards.
Then the real mode handler is called directly, to show the difference
between the two.

Used Interrupt:
get DOS version Int 21h / function 30h
     Input: AH = $30
            AL = $1
     Return: AL = major version number
             AH = minor version number
}

uses
        crt,
        go32;

var
        r : trealregs;
        { temporary variable used for the protected mode int call }
        axreg : Word;

        oldint21h : tseginfo;
        newint21h : tseginfo;

{ this is our int 21h protected mode interupt handler. It catches
the function call to get the DOS version, all other int 21h calls
are redirected to the old handler; it is written in assembly
because the old handler can't be called with pascal }
procedure int21h_handler; assembler;
asm
        cmpw $0x3001, %ax
        jne .LCallOld
        movw $0x3112, %ax
        iret

.LCallOld:
        ljmp %cs:oldint21h
end;

{ a small helper procedure, which waits for a keypress }
procedure resume;
```

```
begin
        Writeln;
        Write('-- press any key to resume --'); readkey;
        gotoxy(1, wherey); clreol;
end;

begin
        { see the text messages for further detail }
        clrscr;
        Writeln('Executing real mode interrupt');
        resume;
        r.ah := $30; r.al := $01;  realintr($21, r);
        Writeln('DOS v', r.al,'.',r.ah, ' detected');
        resume;
        Writeln('Executing protected mode interrupt without our own',
                ' handler');
        Writeln;
        asm
                movb $0x30, %ah
                movb $0x01, %al
                int $0x21
                movw %ax, axreg
        end;
        Writeln('DOS v', r.al,'.',r.ah, ' detected');
        resume;
        Writeln('As you can see the DPMI hosts default protected mode',
                'handler');
        Writeln('simply redirects it to the real mode handler');
        resume;
        Writeln('Now exchanging the protected mode interrupt with our ',
                'own handler');
        resume;

        newint21h.offset := @int21h_handler;
        newint21h.segment := get_cs;
        get_pm_interrupt($21, oldint21h);
        set_pm_interrupt($21, newint21h);

        Writeln('Executing real mode interrupt again');
        resume;
        r.ah := $30; r.al := $01; realintr($21, r);
        Writeln('DOS v', r.al,'.',r.ah, ' detected');
        Writeln;
        Writeln('See, it didn''t change in any way.');
        resume;
        Writeln('Now calling protected mode interrupt');
        resume;
        asm
                movb $0x30, %ah
                movb $0x01, %al
                int $0x21
                movw %ax, axreg
        end;
        Writeln('DOS v', lo(axreg),'.',hi(axreg), ' detected');
        Writeln;
        Writeln('Now you can see that there''s a distinction between ',
                'the two ways of calling interrupts...');
        set_pm_interrupt($21, oldint21h);
```

```
end.
```

54.5 Software interrupts.

Ordinarily, a handler installed with set_pm_interrupt (844) only services software interrupts that are executed in protected mode; real mode software interrupts can be redirected by set_rm_interrupt (846).

See also: set_rm_interrupt (846), get_rm_interrupt (830), set_pm_interrupt (844), get_pm_interrupt (825), lock_data (837), lock_code (837), enable (820), disable (818), outportb (838)

54.6 Hardware interrupts.

Hardware interrupts are generated by hardware devices when something unusual happens; this could be a keypress or a mouse move or any other action. This is done to minimize CPU time, else the CPU would have to check all installed hardware for data in a big loop (this method is called 'polling') and this would take much time. A standard IBM-PC has two interrupt controllers, that are responsible for these hardware interrupts: both allow up to 8 different interrupt sources (IRQs, interrupt requests). The second controller is connected to the first through IRQ 2 for compatibility reasons, e.g. if controller 1 gets an IRQ 2, he hands the IRQ over to controller 2. Because of this up to 15 different hardware interrupt sources can be handled. IRQ 0 through IRQ 7 are mapped to interrupts 8h to Fh and the second controller (IRQ 8 to 15) is mapped to interrupt 70h to 77h. All of the code and data touched by these handlers MUST be locked (via the various locking functions) to avoid page faults at interrupt time. Because hardware interrupts are called (as in real mode) with interrupts disabled, the handler has to enable them before it returns to normal program execution. Additionally a hardware interrupt must send an EOI (end of interrupt) command to the responsible controller; this is accomplished by sending the value 20h to port 20h (for the first controller) or A0h (for the second controller). The following example shows how to redirect the keyboard interrupt.

Listing: ./examples/go32ex/keyclick.pp

```
{ This example demonstrates how to chain to a hardware interrupt.

In more detail, it hooks the keyboard interrupt, calls a user
procedure which in this case simply turns the PC speaker on and off.
Then the old interrupt is called.
}

{$ASMMODE ATT}
{$MODE FPC}

uses
        crt,
        go32;

const
        { keyboard is IRQ 1 -> interrupt 9 }
        kbdint = $9;

var
        { holds old PM interrupt handler address }
        oldint9_handler : tseginfo;
        { new PM interrupt handler }
```

```
        newint9_handler : tseginfo;

        { pointer to interrupt handler }
        clickproc : pointer;
        { the data segment selector }
        backupDS : Word; external name '___v2prt0_ds_alias';

{ interrupt handler }
procedure int9_handler; assembler;
asm
        cli
        { save all registers, because we don't know which the compiler
        uses for the called procedure }
        pushl %ds
        pushl %es
        pushl %fs
        pushl %gs
        pushal
        { set up to call a FPC procedure }
        movw %cs:backupDS, %ax
        movw %ax, %ds
        movw %ax, %es
        movw dosmemselector, %ax
        movw %ax, %fs
        { call user procedure }
        call *clickproc
        { restore all registers }
        popal
        popl %gs
        popl %fs
        popl %es
        popl %ds
        { note: in go32v2 mode %cs=%ds=%es !!!}
        ljmp %cs:oldint9_handler { call old handler }
        { we don't need to do anything more, because the old interrupt
        handler does this for us (send EOI command, iret, sti...) }
end;
{ dummy procedure to retrieve exact length of handler, for locking
and unlocking functions  }
procedure int9_dummy; begin end;

{ demo user procedure, simply clicks on every keypress }
procedure clicker;
begin
        sound(500); delay(10); nosound;
end;
{ dummy procedure to retrieve exact length of user procedure for
locking and unlocking functions }
procedure clicker_dummy; begin end;

{ installs our new handler }
procedure install_click;
begin
        clickproc := @clicker;
        { lock used code and data }
        lock_data(clickproc, sizeof(clickproc));
        lock_data(dosmemselector, sizeof(dosmemselector));
```

```
        lock_code(@clicker,
                longint(@clicker_dummy) - longint(@clicker));
        lock_code(@int9_handler,
                longint(@int9_dummy)-longint(@int9_handler));
        { fill in new handler's 48 bit pointer }
        newint9_handler.offset := @int9_handler;
        newint9_handler.segment := get_cs;
        { get old PM interrupt handler }
        get_pm_interrupt(kbdint, oldint9_handler);
        { set the new interrupt handler }
        set_pm_interrupt(kbdint, newint9_handler);
end;

{ deinstalls our interrupt handler }
procedure remove_click;
begin
        { set old handler }
        set_pm_interrupt(kbdint, oldint9_handler);
        { unlock used code & data }
        unlock_data(dosmemselector, sizeof(dosmemselector));
        unlock_data(clickproc, sizeof(clickproc));

        unlock_code(@clicker,
                longint(@clicker_dummy)-longint(@clicker));
        unlock_code(@int9_handler,
                longint(@int9_dummy)-longint(@int9_handler));
end;

var
        ch : char;

begin
        install_click;
        Writeln('Enter any message. Press return when finished');
        while (ch <> #13) do begin
                ch := readkey; write(ch);
        end;
        remove_click;
end.
```

54.7 Disabling interrupts.

The GO32 unit provides the two procedures disable() and enable() to disable and enable all interrupts.

54.8 Creating your own interrupt handlers.

Interrupt redirection with FPC pascal is done via the set_pm_interrupt() for protected mode interrupts or via the set_rm_interrupt() for real mode interrupts.

54.9 Protected mode interrupts vs. Real mode interrupts.

As mentioned before, there's a distinction between real mode interrupts and protected mode interrupts; the latter are protected mode programs, while the former must be real mode programs. To call a protected mode interrupt handler, an assembly 'int' call must be issued, while the other is called via the realintr() or intr() function. Consequently, a real mode interrupt then must either reside in dos memory (<1MB) or the application must allocate a real mode callback address via the get_rm_callback() function.

54.10 Handling interrupts with DPMI.

The interrupt functions are real-mode procedures; they normally can't be called in protected mode without the risk of an protection fault. So the DPMI host creates an interrupt descriptor table for the application. Initially all software interrupts (except for int 31h, 2Fh and 21h function 4Ch) or external hardware interrupts are simply directed to a handler that reflects the interrupt in real-mode, i.e. the DPMI host's default handlers switch the CPU to real-mode, issue the interrupt and switch back to protected mode. The contents of general registers and flags are passed to the real mode handler and the modified registers and flags are returned to the protected mode handler. Segment registers and stack pointer are not passed between modes.

54.11 Interrupt redirection.

Interrupts are program interruption requests, which in one or another way get to the processor; there's a distinction between software and hardware interrupts. The former are explicitly called by an 'int' instruction and are a bit comparable to normal functions. Hardware interrupts come from external devices like the keyboard or mouse. Functions that handle hardware interrupts are called handlers.

54.12 Processor access.

These are some functions to access various segment registers (%cs, %ds, %ss) which makes your work a bit easier.

See also: get_cs (821), get_ds (822), get_ss (832)

54.13 I/O port access.

The I/O port access is done via the various inportb (836), outportb (838) functions which are available. Additionally Free Pascal supports the Turbo Pascal PORT[]-arrays but it is by no means recommended to use them, because they're only for compatibility purposes.

See also: outportb (838), inportb (836)

54.14 dos memory access.

Dos memory is accessed by the predefined `dosmemselector` selector; the GO32 unit additionally provides some functions to help you with standard tasks, like copying memory from heap to dos memory and the likes. Because of this it is strongly recommended to use them, but you are still free

to use the provided standard memory accessing functions which use 48 bit pointers. The third, but only thought for compatibility purposes, is using the `mem[]`-arrays. These arrays map the whole 1 Mb dos space. They shouldn't be used within new programs. To convert a segment:offset real mode address to a protected mode linear address you have to multiply the segment by 16 and add its offset. This linear address can be used in combination with the DOSMEMSELECTOR variable.

See also: dosmemget (809), dosmemput (810), dosmemmove (810), dosmemfillchar (809), dosmemfillword (809), seg_move (843), seg_fillchar (841), seg_fillword (842)

54.15 FPC specialities.

The %ds and %es selector MUST always contain the same value or some system routines may crash when called. The %fs selector is preloaded with the DOSMEMSELECTOR variable at startup, and it MUST be restored after use, because again FPC relies on this for some functions. Luckily we asm programmers can still use the %gs selector for our own purposes, but for how long ?

See also: get_cs (821), get_ds (822), get_ss (832), allocate_ldt_descriptors (813), free_ldt_descriptor (820), segment_to_descriptor (841), get_next_selector_increment_value (824), get_segment_base_address (832), set_segment_base_address (846), set_segment_limit (846), create_code_segment_alias_descriptor (818)

54.16 Selectors and descriptors.

Descriptors are a bit like real mode segments; they describe (as the name implies) a memory area in protected mode. A descriptor contains information about segment length, its base address and the attributes of it (i.e. type, access rights, ...). These descriptors are stored internally in a so-called descriptor table, which is basically an array of such descriptors. Selectors are roughly an index into this table. Because these 'segments' can be up to 4 GB in size, 32 bits aren't sufficient anymore to describe a single memory location like in real mode. 48 bits are now needed to do this, a 32 bit address and a 16 bit sized selector. The GO32 unit provides the tseginfo record to store such a pointer. But due to the fact that most of the time data is stored and accessed in the %ds selector, FPC assumes that all pointers point to a memory location of this selector. So a single pointer is still only 32 bits in size. This value represents the offset from the data segment base address to this memory location.

54.17 What is DPMI.

The dos Protected Mode Interface helps you with various aspects of protected mode programming. These are roughly divided into descriptor handling, access to dos memory, management of interrupts and exceptions, calls to real mode functions and other stuff. Additionally it automatically provides swapping to disk for memory intensive applications. A DPMI host (either a Windows dos box or CWSDPMI.EXE) provides these functions for your programs.

54.18 Constants, types and variables

54.18.1 Constants

```
auxcarryflag = $010
```

Check for auxiliary carry flag in trealregs (813).

```
carryflag = $001
```

Check for carry flag in trealregs (813).

```
directionflag = $400
```

Check for direction flag in trealregs (813).

```
dosmemfillchar : procedure(seg: Word; ofs: Word; count: LongInt; c
  : Char) = @ dpmi_dosmemfillchar
```

Sets a region of dos memory to a specific byte value.
Parameters:

seg real mode segment.

ofs real mode offset.

count number of bytes to set.

c value to set memory to.

Notes: No range check is performed.

```
dosmemfillword : procedure(seg: Word; ofs: Word; count: LongInt; w
  : Word) = @ dpmi_dosmemfillword
```

Sets a region of dos memory to a specific word value.
Parameters:

seg real mode segment.

ofs real mode offset.

count number of words to set.

w value to set memory to.

Notes: No range check is performed.

```
dosmemget : procedure(seg: Word; ofs: Word; var data; count: LongInt
  ) = @ dpmi_dosmemget
```

Copies data from the dos memory onto the heap.
Parameters:

seg source real mode segment.

ofs source real mode offset.

data destination.

count number of bytes to copy.

Notes: No range checking is performed.
For an example, see global_dos_alloc (833).

```
dosmemmove : procedure(sseg: Word; sofs: Word; dseg: Word; dofs: Word
  ;
          count: LongInt) = @ dpmi_dosmemmove
```

Copies count bytes of data between two dos real mode memory locations.
Parameters:

sseg source real mode segment.

sofs source real mode offset.

dseg destination real mode segment.

dofs destination real mode offset.

count number of bytes to copy.

Notes: No range check is performed in any way.

```
dosmemput : procedure(seg: Word; ofs: Word; var data; count: LongInt
  ) = @ dpmi_dosmemput
```

Copies heap data to dos real mode memory.
Parameters:

seg destination real mode segment.

ofs destination real mode offset.

data source.

count number of bytes to copy.

Notes: No range checking is performed.
For an example, see global_dos_alloc (833).

```
interruptflag = $200
```

Check for interrupt flag in trealregs (813).

```
overflowflag = $800
```

Check for overflow flag in trealregs (813).

```
parityflag = $004
```

Check for parity flag in trealregs (813).

```
rm_dpmi = 4
```

get_run_mode (831) return value: DPMI (e.g. dos box or 386Max).

```
rm_raw = 1
```

get_run_mode (831) return value: raw (without HIMEM).

```
rm_unknown = 0
```

get_run_mode (831) return value: Unknown runmode.

```
rm_vcpi = 3
```

get_run_mode (831) return value: VCPI (with HIMEM and EMM386).

```
rm_xms = 2
```

get_run_mode (831) return value: XMS (with HIMEM, without EMM386).

```
signflag = $080
```

Check for sign flag in trealregs (813).

```
trapflag = $100
```

Check for trap flag in trealregs (813).

```
zeroflag = $040
```

Check for zero flag in trealregs (813).

54.18.2 Types

```
registers = trealregs
```

Alias for trealregs (813).

```
trealregs = record
case Integer of
1: (
  EDI : LongInt;
  ESI : LongInt
  ;
  EBP : LongInt;
  Res : LongInt;
  EBX : LongInt;
  EDX : LongInt
  ;
  ECX : LongInt;
  EAX : LongInt;
```

```
  Flags : Word;
  ES : Word;
  DS : Word;
  FS : Word;
  GS : Word;
  IP : Word;
  CS : Word;
  SP : Word;
  SS : Word;
);
2: (
  DI : Word;
  DI2 : Word;
  SI
  : Word;
  SI2 : Word;
  BP : Word;
  BP2 : Word;
  R1 : Word;
  R2
  : Word;
  BX : Word;
  BX2 : Word;
  DX : Word;
  DX2 : Word;
  CX
  : Word;
  CX2 : Word;
  AX : Word;
  AX2 : Word;
);
3: (
  stuff
  : Array[1..4] of LongInt;
  BL : Byte;
  BH : Byte;
  BL2 : Byte
  ;
  BH2 : Byte;
  DL : Byte;
  DH : Byte;
  DL2 : Byte;
  DH2 : Byte
  ;
  CL : Byte;
  CH : Byte;
  CL2 : Byte;
  CH2 : Byte;
  AL : Byte
  ;
  AH : Byte;
  AL2 : Byte;
  AH2 : Byte;
);
```

```
4: (
  RealEDI : LongInt
  ;
  RealESI : LongInt;
  RealEBP : LongInt;
  RealRES : LongInt;
  RealEBX : LongInt;
  RealEDX : LongInt;
  RealECX : LongInt;
  RealEAX
  : LongInt;
  RealFlags : Word;
  RealES : Word;
  RealDS : Word;
  RealFS : Word;
  RealGS : Word;
  RealIP : Word;
  RealCS : Word
  ;
  RealSP : Word;
  RealSS : Word;
);
end
```

The `trealregs` type contains the data structure to pass register values to a interrupt handler or real mode callback.

54.18.3 Variables

```
dosmemselector : Word
```

Selector to the dos memory. The whole dos memory is automatically mapped to this single descriptor at startup. This selector is the recommended way to access dos memory.

```
int31error : Word
```

This variable holds the result of a DPMI interrupt call. Any nonzero value must be treated as a critical failure.

54.19 Procedures and functions

54.19.1 allocate_ldt_descriptors

Synopsis: Allocate a number of descriptors.

Declaration: `function allocate_ldt_descriptors(count: Word) : Word`

Visibility: default

Description: Allocates a number of new descriptors.

Parameters:

count:\specifies the number of requested unique descriptors.

Return value: The base selector.

Remark Notes: The descriptors allocated must be initialized by the application with other function calls. This function returns descriptors with a limit and size value set to zero. If more than one descriptor was requested, the function returns a base selector referencing the first of a contiguous array of descriptors. The selector values for subsequent descriptors in the array can be calculated by adding the value returned by the get_next_selector_increment_value (824) function.

Errors: Check the int31error (813) variable.

See also: free_ldt_descriptor (820), get_next_selector_increment_value (824), segment_to_descriptor (841), create_code_segment_alias_descriptor (818), set_segment_limit (846), set_segment_base_address (846)

Listing: ./examples/go32ex/seldes.pp

```
{
This example demonstrates the usage of descriptors and the effects of
changing its limit and base address.

In more detail, the program fills the region described by an
allocated descriptor in text screen memory with various characters.
Before doing this it saves the entire screen contents to the heap and
restores it afterwards.

Some additional background:

The text screen of a VGA card has it's address space at $B800:0;
screen memory is organized in a linear fashion, e.g. the second line
comes directly after the first, where each cell occupies 2 bytes of
memory (1 byte character data, 1 byte attributes). It is 32 kb in
size.

Hence the offset of a single memory cell from its origin is:

        Y * columns * 2 + X * 2

where X and Y mark the point and columns is the number of character
cells per line
}
{$mode delphi}
uses
        crt,
        go32;

const
        { screen x and y dimensions }
        maxx = 80;
        maxy = 25;
        { bytes used for every character cell }
        bytespercell = 2;
        { screen size in bytes }
        screensize = maxx * maxy * bytespercell;

        { the linear address of $B800:0 }
        linB8000 = $B800 * 16;

type
        string80 = string[80];
```

```
var
        { holds the old screen contents }
        text_save : array[0..screensize-1] of byte;
        { old cursor x and y coordinates }
        text_oldx, text_oldy : Word;

        { selector to the text mode screen }
        text_sel : Word;

{ prints a status message on the first line of the screen and then
waits for a keypress }
procedure status(s : string80);
begin
     gotoxy(1, 1); clreol; write(s); readkey;
end;

{ writes some descriptor info on the last 2 lines }
procedure selinfo(sel : Word);
begin
     gotoxy(1, 24);
     clreol; writeln('Descriptor base address : $',
        hexstr(get_segment_base_address(sel), 8));
     clreol; write('Descriptor limit : ', get_segment_limit(sel));
end;

{ returns a 2 byte character cell, which includes character data
and its color attributes }
function makechar(ch : char; color : byte) : Word;
begin
     result := byte(ch) or (color shl 8);
end;

begin
     { save original screen contents to variable, this time by using
     seg_move() and the dosmemselector variable }
     seg_move(dosmemselector, linB8000, get_ds, longint(@text_save),
        screensize);
     { additionally we have to save the old screen cursor
     coordinates }
     text_oldx := wherex; text_oldy := wherey;
     { clear the whole screen }
     seg_fillword(dosmemselector, linB8000, screensize div 2,
        makechar(' ', Black or (Black shl 4)));
     { output message }
     status('Creating selector ''text_sel'' to a part of ' +
        'text screen memory');
     { allocate descriptor }
     text_sel := allocate_ldt_descriptors(1);
     { set its base address to the linear address of the text screen
     + the byte size of one line (=maxx * bytespercell * 1) }
     set_segment_base_address(text_sel,
        linB8000 + bytespercell * maxx * 1);
     { the limit is set to the screensize reduced by one (a must be)
     and the number of lines we don't want to have touched (first
     line + lower 2 lines) }
     set_segment_limit(text_sel, screensize - 1 - bytespercell *
        maxx * 3);
```

```
    { write descriptor info  }
    selinfo(text_sel);

    status('and clearing entire memory selected by ''text_sel''' +
       ' descriptor');
    { fill the entire selected memory with single characters }
    seg_fillword(text_sel, 0, (get_segment_limit(text_sel)+1) div 2,
       makechar(' ', LightBlue shl 4));

    status('Notice that only the memory described by the ' +
       'descriptor changed, nothing else');

    status('Now reducing it''s limit and base and setting it''s ' +
       'described memory');
    { set the base address of the descriptor (increase it by the
    byte size of one line) }
    set_segment_base_address(text_sel,
       get_segment_base_address(text_sel) + bytespercell * maxx);
    { decrease the limit by byte size of 2 lines (1 line because
       base address changed, one line on the lower end) }
    set_segment_limit(text_sel,
       get_segment_limit(text_sel) - bytespercell * maxx * 2);
    { write descriptor info  }
    selinfo(text_sel);
    status('Notice that the base addr increased by one line but ' +
       'the limit decreased by 2 lines');
    status('This should give you the hint that the limit is ' +
       'relative to the base');
    { fill the descriptor area }
    seg_fillword(text_sel, 0, (get_segment_limit(text_sel)+1) div 2,
       makechar(#176, LightMagenta or Brown shl 4));

    status('Now let''s get crazy and copy 10 lines of data from ' +
       'the previously saved screen');
    { copy memory from the data segment to screen }
    seg_move(get_ds, longint(@text_save), text_sel,
       maxx * bytespercell * 2, maxx * bytespercell * 10);

    status('At last freeing the descriptor and restoring the old ' +
       ' screen contents..');
    status('I hope this little program may give you some hints ' +
       'on working with descriptors');
    { free the descriptor so that it can be used for things }
    free_ldt_descriptor(text_sel);
    { restore old state  }
    seg_move(get_ds, longint(@text_save), dosmemselector,
       linB8000, screensize);
    gotoxy(text_oldx, text_oldy);
end.
```

54.19.2 allocate_memory_block

Synopsis: Allocate a block of linear memory.

Declaration: `function allocate_memory_block(size: LongInt) : LongInt`

Visibility: default

Description: Allocates a block of linear memory.

Parameters:

size: Size of requested linear memory block in bytes.

Returned values: blockhandle - the memory handle to this memory block. Linear address of the requested memory.

Remark *warning* According to my DPMI docs this function is not implemented correctly. Normally you should also get a blockhandle to this block after successful operation. This handle can then be used to free the memory block afterwards or use this handle for other purposes. Since the function isn't implemented correctly, and doesn't return a blockhandle, the block can't be deallocated and is hence unusable ! This function doesn't allocate any descriptors for this block, it's the applications responsibility to allocate and initialize for accessing this memory.

Errors: Check the int31error (813) variable.

See also: free_memory_block (820)

54.19.3 copyfromdos

Synopsis: Copy data from DOS to heap.

Declaration: `procedure copyfromdos(var addr; len: LongInt)`

Visibility: default

Description: Copies data from the pre-allocated dos memory transfer buffer to the heap.

Parameters:

addrdata to copy to.

lennumber of bytes to copy to heap.

Notes: Can only be used in conjunction with the dos memory transfer buffer.

Errors: Check the int31error (813) variable.

See also: tb_size (847), transfer_buffer (848), copytodos (817)

54.19.4 copytodos

Synopsis: Copy data from heap to DOS memory.

Declaration: `procedure copytodos(var addr; len: LongInt)`

Visibility: default

Description: Copies data from heap to the pre-allocated dos memory buffer.

Parameters:

addrdata to copy from.

lennumber of bytes to copy to dos memory buffer.

Notes: This function fails if you try to copy more bytes than the transfer buffer is in size. It can only be used in conjunction with the transfer buffer.

Errors: Check the int31error (813) variable.

See also: tb_size (847), transfer_buffer (848), copyfromdos (817)

54.19.5 create_code_segment_alias_descriptor

Synopsis: Create new descriptor from existing descriptor.

Declaration: `function create_code_segment_alias_descriptor(seg: Word) : Word`

Visibility: default

Description: Creates a new descriptor that has the same base and limit as the specified descriptor.

Parameters:

segDescriptor.

Return values: The data selector (alias).

Notes: In effect, the function returns a copy of the descriptor. The descriptor alias returned by this function will not track changes to the original descriptor. In other words, if an alias is created with this function, and the base or limit of the original segment is then changed, the two descriptors will no longer map the same memory.

Errors: Check the int31error (813) variable.

See also: allocate_ldt_descriptors (813), set_segment_limit (846), set_segment_base_address (846)

54.19.6 disable

Synopsis: Disable hardware interrupts.

Declaration: `procedure disable`

Visibility: default

Description: Disables all hardware interrupts by execution a CLI instruction.

Errors: None.

See also: enable (820)

54.19.7 dpmi_dosmemfillchar

Synopsis: Fill DOS memory with a character.

Declaration:
```
procedure dpmi_dosmemfillchar(seg: Word; ofs: Word; count: LongInt;
                              c: Char)
```

Visibility: default

Description: `dpmi_dosmemfillchar` fills the DOS memory region indicated by `seg`,`ofs` with `count` characters `c`.

See also: dpmi_dosmemput (819), dpmi_dosmemget (819), dpmi_dosmemmove (819), dpmi_dosmemfillword (819)

54.19.8 dpmi_dosmemfillword

Synopsis: Fill DOS memory with a word value.

Declaration:
```
procedure dpmi_dosmemfillword(seg: Word; ofs: Word; count: LongInt;
                              w: Word)
```

Visibility: default

Description: `dpmi_dosmemfillword` fills the DOS memory region indicated by `seg`,`ofs` with `count` words `W`.

See also: dpmi_dosmemput (819), dpmi_dosmemget (819), dpmi_dosmemfillchar (818), dpmi_dosmemmove (819)

54.19.9 dpmi_dosmemget

Synopsis: Move data from DOS memory to DPMI memory.

Declaration:
```
procedure dpmi_dosmemget(seg: Word; ofs: Word; var data; count: LongInt)
```

Visibility: default

Description: `dpmi_dosmemput` moves `count` bytes of data from the DOS memory location indicated by `seg` and `ofs` to DPMI memory indicated by `data`.

See also: dpmi_dosmemput (819), dpmi_dosmemmove (819), dpmi_dosmemfillchar (818), dpmi_dosmemfillword (819)

54.19.10 dpmi_dosmemmove

Synopsis: Move DOS memory.

Declaration:
```
procedure dpmi_dosmemmove(sseg: Word; sofs: Word; dseg: Word;
                          dofs: Word; count: LongInt)
```

Visibility: default

Description: `dpmi_dosmemmove` moves `count` bytes from DOS memory `sseg`,`sofs` to `dseg`,`dofs`.

See also: dpmi_dosmemput (819), dpmi_dosmemget (819), dpmi_dosmemfillchar (818), dpmi_dosmemfillword (819)

54.19.11 dpmi_dosmemput

Synopsis: Move data from DPMI memory to DOS memory.

Declaration:
```
procedure dpmi_dosmemput(seg: Word; ofs: Word; var data; count: LongInt)
```

Visibility: default

Description: `dpmi_dosmemput` moves `count` bytes of data from `data` to the DOS memory location indicated by `seg` and `ofs`.

See also: dpmi_dosmemget (819), dpmi_dosmemmove (819), dpmi_dosmemfillchar (818), dpmi_dosmemfillword (819)

54.19.12 enable

Synopsis: Enable hardware interrupts.

Declaration: `procedure enable`

Visibility: default

Description: Enables all hardware interrupts by executing a STI instruction.

Errors: None.

See also: disable (818)

54.19.13 free_ldt_descriptor

Synopsis: Free a descriptor.

Declaration: `function free_ldt_descriptor(d: Word) : Boolean`

Visibility: default

Description: Frees a previously allocated descriptor.

Parameters:

desThe descriptor to be freed.

Return value: `True` if successful, `False` otherwise. Notes: After this call this selector is invalid and must not be used for any memory operations anymore. Each descriptor allocated with allocate_ldt_descriptors (813) must be freed individually with this function, even if it was previously allocated as a part of a contiguous array of descriptors.

For an example, see allocate_ldt_descriptors (813).

Errors: Check the int31error (813) variable.

See also: allocate_ldt_descriptors (813), get_next_selector_increment_value (824)

54.19.14 free_linear_addr_mapping

Synopsis: ? No description available.

Declaration: `function free_linear_addr_mapping(linear_addr: DWord) : Boolean`

Visibility: default

54.19.15 free_memory_block

Synopsis: Free allocated memory block.

Declaration: `function free_memory_block(blockhandle: LongInt) : Boolean`

Visibility: default

Description: Frees a previously allocated memory block.

Parameters:

blockhandlethe handle to the memory area to free.

Return value: `True` if successful, `false` otherwise. Notes: Frees memory that was previously allocated with allocate_memory_block (816) . This function doesn't free any descriptors mapped to this block, it's the application's responsibility.

Errors: Check int31error (813) variable.

See also: allocate_memory_block (816)

54.19.16 free_rm_callback

Synopsis: Release real mode callback.

Declaration: `function free_rm_callback(var intaddr: tseginfo) : Boolean`

Visibility: default

Description: Releases a real mode callback address that was previously allocated with the get_rm_callback (826) function.

Parameters:

intaddrreal mode address buffer returned by get_rm_callback (826) .

Return values: `True` if successful, `False` if not

For an example, see get_rm_callback (826).

Errors: Check the int31error (813) variable.

See also: set_rm_interrupt (846), get_rm_callback (826)

54.19.17 get_cs

Synopsis: Get CS selector.

Declaration: `function get_cs : Word`

Visibility: default

Description: Returns the cs selector.

Return value: The content of the cs segment register.

For an example, see set_pm_interrupt (844).

Errors: None.

See also: get_ds (822), get_ss (832)

54.19.18 get_descriptor_access_right

Synopsis: Get descriptor's access rights.

Declaration: `function get_descriptor_access_right(d: Word) : LongInt`

Visibility: default

Description: Gets the access rights of a descriptor.

Parameters:

dselector to descriptor.

Return value: Access rights bit field.

Errors: Check the int31error (813) variable.

See also: set_descriptor_access_right (843)

54.19.19 get_dpmi_version

Synopsis: Return DPMI information.

Declaration: `function get_dpmi_version(var version: tdpmiversioninfo) : Boolean`

Visibility: default

Description: `get_dpmi_version` returns version information (Int $31 Function $0400) in `Version` and returns `True` if the information was retrieved successfully, `false` if the call failed.

Errors: The call returns `false` if the information could not be retrieved.

See also: tdpmiversioninfo (849)

54.19.20 get_ds

Synopsis: Get DS Selector.

Declaration: `function get_ds : Word`

Visibility: default

Description: Returns the ds selector.

Return values: The content of the ds segment register.

Errors: None.

See also: get_cs (821), get_ss (832)

54.19.21 get_exception_handler

Synopsis: Return current exception handler.

Declaration: `function get_exception_handler(e: Byte; var intaddr: tseginfo) : Boolean`

Visibility: default

Description: `get_exception_handler` returns the exception handler for exception `E` in `intaddr`. It returns `True` if the call was successful, `False` if not.

See also: set_exception_handler (843), get_pm_exception_handler (825)

54.19.22 get_linear_addr

Synopsis: Convert physical to linear address.

Declaration: `function get_linear_addr(phys_addr: DWord; size: LongInt) : DWord`

Visibility: default

Description: Converts a physical address into a linear address.

Parameters:

phys_addrphysical address of device.

sizeSize of region to map in bytes.

Return value: Linear address that can be used to access the physical memory. Notes: It's the applications responsibility to allocate and set up a descriptor for access to the memory. This function shouldn't be used to map real mode addresses.

Errors: Check the int31error (813) variable.

See also: allocate_ldt_descriptors (813), set_segment_limit (846), set_segment_base_address (846)

54.19.23 get_meminfo

Synopsis: Return information on the available memory.

Declaration: `function get_meminfo(var meminfo: tmeminfo) : Boolean`

Visibility: default

Description: Returns information about the amount of available physical memory, linear address space, and disk space for page swapping.

Parameters:

meminfobuffer to fill memory information into.

Return values: Due to an implementation bug this function always returns `False`, but it always succeeds.

Remark Notes: Only the first field of the returned structure is guaranteed to contain a valid value. Any fields that are not supported by the DPMI host will be set by the host to `-1 (0FFFFFFFFH)` to indicate that the information is not available. The size of the pages used by the DPMI host can be obtained with the get_page_size (825) function.

Errors: Check the int31error (813) variable.

See also: get_page_size (825)

Listing: ./examples/go32ex/meminfo.pp

```
{ Shows how to obtain memory information via get_meminfo();

notice the checks if any of the returned information is invalid (-1)
}

uses
        go32;

var
```

```
        meminfo : tmeminfo;

begin
        get_meminfo(meminfo);
        if (int31error <> 0) then begin
                Writeln('Error getting DPMI memory information... Halting');
                Writeln('DPMI error number : ', int31error);
        end else begin
                with meminfo do begin
                        Writeln('Largest available free block : ',
                                available_memory div 1024, ' kbytes');
                        if (available_pages <> -1) then
                                Writeln('Maximum available unlocked pages : ',
                                        available_pages);
                        if (available_lockable_pages <> -1) then
                                Writeln('Maximum lockable available pages : ',
                                        available_lockable_pages);
                        if (linear_space <> -1) then
                                Writeln('Linear address space size : ',
                                        linear_space*get_page_size div 1024, ' kbytes');
                        if (unlocked_pages <> -1) then
                                Writeln('Total number of unlocked pages : ',
                                        unlocked_pages);
                        if (available_physical_pages <> -1) then
                                Writeln('Total number of free pages : ',
                                        available_physical_pages);
                        if (total_physical_pages <> -1) then
                                Writeln('Total number of physical pages : ',
                                        total_physical_pages);
                        if (free_linear_space <> -1) then
                                Writeln('Free linear address space : ',
                                        free_linear_space*get_page_size div 1024,
                                        ' kbytes');
                        if (max_pages_in_paging_file <> -1) then
                                Writeln('Maximum size of paging file : ',
                                        max_pages_in_paging_file*get_page_size div 1024,
                                        ' kbytes');
                end;
        end;
end.
```

54.19.24 get_next_selector_increment_value

Synopsis: Return selector increment value.

Declaration: `function get_next_selector_increment_value : Word`

Visibility: default

Description: Returns the selector increment value when allocating multiple subsequent descriptors via allocate_ldt_descriptors (813).

Return value: Selector increment value.

Remark Notes: Because allocate_ldt_descriptors (813) only returns the selector for the first descriptor and so the value returned by this function can be used to calculate the selectors for subsequent descriptors in the array.

Errors: Check the int31error (813) variable.

See also: allocate_ldt_descriptors (813), free_ldt_descriptor (820)

54.19.25 get_page_attributes

Synopsis: ? No description available.

Declaration:
```
function get_page_attributes(handle: DWord; offset: DWord;
                             pagecount: DWord; buf: pointer) : Boolean
```

Visibility: default

54.19.26 get_page_size

Synopsis: Return the page size.

Declaration:
```
function get_page_size : LongInt
```

Visibility: default

Description: Returns the size of a single memory page.

Return value: Size of a single page in bytes.

Remark The returned size is typically 4096 bytes.

For an example, see get_meminfo (823).

Errors: Check the int31error (813) variable.

See also: get_meminfo (823)

54.19.27 get_pm_exception_handler

Synopsis: Get protected mode exception handler.

Declaration:
```
function get_pm_exception_handler(e: Byte; var intaddr: tseginfo)
                                  : Boolean
```

Visibility: default

Description: `get_pm_exception_handler` returns the protected mode exception handler for exception `E` in `intaddr`. It returns `True` if the call was successful, `False` if not.

See also: get_exception_handler (822), set_pm_exception_handler (844)

54.19.28 get_pm_interrupt

Synopsis: Return protected mode interrupt handler.

Declaration:
```
function get_pm_interrupt(vector: Byte; var intaddr: tseginfo) : Boolean
```

Visibility: default

Description: Returns the address of a current protected mode interrupt handler.

Parameters:

vectorinterrupt handler number you want the address to.

intaddrbuffer to store address.

Return values: `True` if successful, `False` if not.

Remark The returned address is a protected mode selector:offset address.

For an example, see set_pm_interrupt (844).

Errors: Check the int31error (813) variable.

See also: set_pm_interrupt (844), set_rm_interrupt (846), get_rm_interrupt (830)

54.19.29 get_rm_callback

Synopsis: Return real mode callback.

Declaration:
```
function get_rm_callback(pm_func: pointer; const reg: trealregs;
                         var rmcb: tseginfo) : Boolean
```

Visibility: default

Description: Returns a unique real mode `segment:offset` address, known as a "real mode callback," that will transfer control from real mode to a protected mode procedure.

Parameters:

pm_funcpointer to the protected mode callback function.

regsupplied registers structure.

rmcbbuffer to real mode address of callback function.

Return values: `True` if successful, otherwise `False`.

Remark Callback addresses obtained with this function can be passed by a protected mode program for example to an interrupt handler, device driver, or TSR, so that the real mode program can call procedures within the protected mode program or notify the protected mode program of an event. The contents of the supplied regs structure is not valid after function call, but only at the time of the actual callback.

Errors: Check the int31error (813) variable.

See also: free_rm_callback (821)

Listing: ./examples/go32ex/callback.pp

```
{ This program tries to give an example how to install a callback
procedure with the help of the GO32 unit.

It installs a callback which is supplied by any Microsoft compatible
mouse driver; at a specified mouse action this routine is called.
This callback must provide the services explained in the docs. The
main callback has to be in assembly, because it isn't possible to do
these services with pascal alone. But is written as general as
possible to provide maximum re-usability for other applications and
hence it simply calls a normal pascal user procedure in addition to
some initialization and callback service code, so you don't need to
hassle around with it too much.

Notes to this user procedure :
*) it should not last too long to execute it
*) ALL data and code touched in this proc MUST be locked BEFORE it is
called the first time
```

```
Used software interrupt calls (rough descriptions, only what's used):

Int 33h 0000h - Microsoft Mouse driver : Reset mouse
Input : AX = 0000h
Return : AX = FFFFh if successful
         BX = number of buttons (if FFFFh then mouse has 2 buttons)

Int 33h 0001h - Microsoft Mouse driver : Show mouse cursor
Input : AX = 0001h
Return : Mouse cursor shown on screen

Int 33h 0002h - Microsoft mouse driver : Hide mouse cursor
Input : AX = 0002h
Return : Hides mouse cursor again

 Int 33h 000Ch - Microsoft mouse driver : Install user callback
Input : AX = 000Ch
        CX = bit mask which tells the mouse driver at which actions
        the callback should be called, i.e. if button pressed, mouse
        moved etc.
        (In this example it's set to 7Fh so that the callback is
        called on every action)
        ES:EDX = pointer to callback procedure to call
 Note : The registers structure supplied to the callback contains
        valid mouse data when the handler is called.
  BX = button state information
  CX = mouse X coordinates
  DX = mouse Y coordinates
For more detailed information consult any mouse reference or
interrupt list.
}
{$CALLING REGISTER}
{$ASMMODE ATT}
{$MODE FPC}

uses
        crt,
        go32;

const
        { the mouse interrupt number }
        mouseint = $33;

var
        { supplied register structure to the callback }
        mouse_regs     : trealregs; external name '___v2prt0_rmcb_regs';
        { real mode 48 bit pointer to the callback }
        mouse_seginfo : tseginfo;

var
        { number of mouse buttons }
        mouse_numbuttons : longint;

        { bit mask for the action which triggered the callback }
        mouse_action : word;
        { current mouse x and y coordinates }
        mouse_x, mouse_y : Word;
```

```
        { button state }
        mouse_b : Word;

        { is an additional user procedure installed }
        userproc_installed : Longbool;
        { length of additional user procedure }
        userproc_length : Longint;
        { pointer to user proc }
        userproc_proc : pointer;

{ callback control handler, calls a user procedure if installed }

{ callback control handler, calls a user procedure if installed }
procedure callback_handler; assembler;
asm
   pushw %ds
   pushl %eax
   movw %es, %ax
   movw %ax, %ds

   { give control to user procedure if installed }
   cmpl $0, USERPROC_INSTALLED
   je .LNoCallback
   pushal
   movw DOSmemSELECTOR, %ax
   movw %ax, %fs  { set fs for FPC }
   call *USERPROC_PROC
   popal
.LNoCallback:

   popl %eax
   popw %ds

   pushl %eax
   movl (%esi), %eax
   movl %eax, %es: 42(%edi) { adjust stack }
   addw $4, %es:46(%edi)
   popl %eax
   iret
end;
{ This dummy is used to obtain the length of the callback control
function. It has to be right after the callback_handler() function.
}
procedure mouse_dummy; begin end;

{ This is the supplied user procedure. In this case we simply
transform the virtual 640x200 mouse coordinate system to a 80x25
text mode coordinate system }
procedure textuserproc;
begin
        { the mouse_regs record contains the real mode registers now }
        mouse_b := mouse_regs.bx;
        mouse_x := (mouse_regs.cx shr 3) + 1;
        mouse_y := (mouse_regs.dx shr 3) + 1;
end;

{ Description : Installs the mouse callback control handler and
handles all necessary mouse related initialization.
```

```
  Input : userproc - pointer to a user procedure, nil if none
          userproclen - length of user procedure
}
procedure install_mouse(userproc : pointer; userproclen : longint);
var r : trealregs;
begin
        { mouse driver reset }
        r.eax := $0; realintr(mouseint, r);
        if (r.eax <> $FFFF) then begin
                Writeln('No Microsoft compatible mouse found');
                Writeln('A Microsoft compatible mouse driver is necessary ',
                        'to run this example');
                halt;
        end;
        { obtain number of mouse buttons }
        if (r.bx = $ffff) then mouse_numbuttons := 2
        else mouse_numbuttons := r.bx;
        Writeln(mouse_numbuttons, ' button Microsoft compatible mouse ',
                ' found.');
        { check for additional user procedure, and install it if
        available }
        if (userproc <> nil) then begin
                userproc_proc := userproc;
                userproc_installed := true;
                userproc_length := userproclen;
                { lock code for user procedure }
                lock_code(userproc_proc, userproc_length);
        end else begin
                { clear variables }
                userproc_proc := nil;
                userproc_length := 0;
                userproc_installed := false;
        end;
        { lock code & data which is touched in the callback handler }
        lock_data(mouse_x, sizeof(mouse_x));
        lock_data(mouse_y, sizeof(mouse_y));
        lock_data(mouse_b, sizeof(mouse_b));
        lock_data(mouse_action, sizeof(mouse_action));

        lock_data(userproc_installed, sizeof(userproc_installed));
        lock_data(userproc_proc, sizeof(userproc_proc));

        lock_data(mouse_regs, sizeof(mouse_regs));
        lock_data(mouse_seginfo, sizeof(mouse_seginfo));
        lock_code(@callback_handler,
                longint(@mouse_dummy)-longint(@callback_handler));
        { allocate callback (supply registers structure) }
        get_rm_callback(@callback_handler, mouse_regs, mouse_seginfo);
        { install callback }
        r.eax := $0c; r.ecx := $7f;
        r.edx := longint(mouse_seginfo.offset);
        r.es := mouse_seginfo.segment;
        realintr(mouseint, r);
        { show mouse cursor }
        r.eax := $01;
        realintr(mouseint, r);
end;
```

```
procedure remove_mouse;
var
        r : trealregs;
begin
        { hide mouse cursor }
        r.eax := $02; realintr(mouseint, r);
        { remove callback handler }
        r.eax := $0c; r.ecx := 0; r.edx := 0; r.es := 0;
        realintr(mouseint, r);
        { free callback }
        free_rm_callback(mouse_seginfo);
        { check if additional userproc is installed, and clean up if
        needed }
        if (userproc_installed) then begin
                unlock_code(userproc_proc, userproc_length);
                userproc_proc := nil;
                userproc_length := 0;
                userproc_installed := false;
        end;
        { unlock used code & data }
        unlock_data(mouse_x, sizeof(mouse_x));
        unlock_data(mouse_y, sizeof(mouse_y));
        unlock_data(mouse_b, sizeof(mouse_b));
        unlock_data(mouse_action, sizeof(mouse_action));

        unlock_data(userproc_proc, sizeof(userproc_proc));
        unlock_data(userproc_installed, sizeof(userproc_installed));

        unlock_data(mouse_regs, sizeof(mouse_regs));
        unlock_data(mouse_seginfo, sizeof(mouse_seginfo));
        unlock_code(@callback_handler,
                longint(@mouse_dummy)-longint(@callback_handler));
        fillchar(mouse_seginfo, sizeof(mouse_seginfo), 0);
end;


begin
        install_mouse(@textuserproc, 400);
        Writeln('Press any key to exit...');
        while (not keypressed) do begin
                { write mouse state info }
                gotoxy(1, wherey);
                write('MouseX : ', mouse_x:2, ' MouseY : ', mouse_y:2,
                        ' Buttons : ', mouse_b:2);
        end;
        remove_mouse;
end.
```

54.19.30 get_rm_interrupt

Synopsis: Get real mode interrupt vector.

Declaration: `function get_rm_interrupt(vector: Byte; var intaddr: tseginfo) : Boolean`

Visibility: default

Description: Returns the contents of the current machine's real mode interrupt vector for the specified interrupt.

Parameters:

vectorinterrupt vector number.

intaddrbuffer to store real mode `segment:offset` address.

Return values: `True` if successful, `False` otherwise.

Remark The returned address is a real mode segment address, which isn't valid in protected mode.

Errors: Check the int31error (813) variable.

See also: set_rm_interrupt (846), set_pm_interrupt (844), get_pm_interrupt (825)

54.19.31 get_run_mode

Synopsis: Return current run mode.

Declaration: `function get_run_mode : Word`

Visibility: default

Description: Returns the current mode your application runs with.

Return values: One of the constants used by this function.

Errors: None.

See also: get_run_mode (831)

Listing: ./examples/go32ex/getrunmd.pp

```
{ Simply write a message according to the current environment }

uses
        go32;

begin
        { depending on the detected environment we simply write
        another message Note: in go32v2 this will always be rm_dpmi. }

        case (get_run_mode) of
                rm_unknown :
                        WriteLn('Unknown environment found');
                rm_raw     :
                        WriteLn('You are currently running in raw mode ',
                                '(without HIMEM)');
                rm_xms     :
                        WriteLn('You are currently using HIMEM.SYS only');
                rm_vcpi    :
                        WriteLn('VCPI server detected. You''re using HIMEM and ',
                                'EMM386');
                rm_dpmi    :
                        WriteLn('DPMI detected. You''re using a DPMI host like ',
                                'a windows DOS box or CWSDPMI');
        end;
end.
```

54.19.32 get_segment_base_address

Synopsis: Return base address from descriptor table.

Declaration: `function get_segment_base_address(d: Word) : DWord`

Visibility: default

Description: Returns the 32-bit linear base address from the descriptor table for the specified segment.

Parameters:

dselector of the descriptor you want the base address of.

Return values: Linear base address of specified descriptor.

For an example, see allocate_ldt_descriptors (813).

Errors: Check the int31error (813) variable.

See also: allocate_ldt_descriptors (813), set_segment_base_address (846), allocate_ldt_descriptors (813), set_segment_limit (846), get_segment_limit (832)

54.19.33 get_segment_limit

Synopsis: Return segment limits from descriptor.

Declaration: `function get_segment_limit(d: Word) : DWord`

Visibility: default

Description: Returns a descriptors segment limit.

Parameters:

dselector.

Return value: Limit of the descriptor in bytes.

Errors: Returns zero if descriptor is invalid.

See also: allocate_ldt_descriptors (813), set_segment_limit (846), set_segment_base_address (846), get_segment_base_address (832)

54.19.34 get_ss

Synopsis: Return SS selector.

Declaration: `function get_ss : Word`

Visibility: default

Description: Returns the ss selector.

Return values: The content of the ss segment register.

Errors: None.

See also: get_ds (822), get_cs (821)

54.19.35 global_dos_alloc

Synopsis: Allocate DOS real mode memory.

Declaration: `function global_dos_alloc(bytes: LongInt) : LongInt`

Visibility: default

Description: Allocates a block of dos real mode memory.

Parameters:

bytessize of requested real mode memory.

Return values: The low word of the returned value contains the selector to the allocated dos memory block, the high word the corresponding real mode segment value. The offset value is always zero. This function allocates memory from dos memory pool, i.e. memory below the 1 MB boundary that is controlled by dos. Such memory blocks are typically used to exchange data with real mode programs, TSRs, or device drivers. The function returns both the real mode segment base address of the block and one descriptor that can be used by protected mode applications to access the block. This function should only used for temporary buffers to get real mode information (e.g. interrupts that need a data structure in ES:(E)DI), because every single block needs an unique selector. The returned selector should only be freed by a global_dos_free (835) call.

Errors: Check the int31error (813) variable.

See also: global_dos_free (835)

Listing: ./examples/go32ex/buffer.pp

```
{ This program demonstrates the usage of DOS real mode memory by
executing a software interrupt which needs a buffer to store data
into. Because these interrupts are real mode funcs, the buffer must
be located in real mode memory space (first MB of memory). Such
memory can only be allocated by the global_dos_alloc() and
global_dos_free() functions of the GO32 unit.

In more detail this program tries to detect a VESA 2.0 BIOS
extension of your graphics card and outputs its version.

Here's the necessary interrupt call description:

  Int 10h 4f00h : VESA BIOS extension installation check
  Input : AX = 4F00h
          ES:DI = pointer to 512 byte information buffer
  Output : AX = 004Fh if successful
           ES:DI = pointer to filled buffer

  Buffer structure : (relevant to this example)

            must be 'VESA' in the first 4 chars of the buffer to be
            valid VBE version in the next word

  Note : to request VBE 2.0 information, the first 4 bytes of the
         buffer must contain 'VBE2' prior to the interrupt call.

         (this makes the problem a bit tougher; we first have to copy the
         buffer with the 'VBE2' id to dos memory...)
}
```

```
uses
        go32;

{The following 2 functions are wrappers to the GO32
global_dos_alloc() and global_dos_free() functions to simplify their
usage }

{ Function : dosalloc }
{ Input    : size of a real mode location }
{ Output   : selector and segment of a real mode location }
procedure dosalloc(var selector : word;
        var segment : word; size : longint);
var
        res : longint;
begin
        { try to allocate real mode memory  }
        res := global_dos_alloc(size);
        { the lower 16 bits of the result contain the selector to the
        allocated memory block }
        selector := word(res);
        { the upper 16 bits contain the real mode segment address of
        this block; the offset is always 0, so we don't need to return
        this }
        segment := word(res shr 16);
end;

{ Function    : dosfree }
{ Input       : selector of a real mode block }
{ Output      : none }
{ Description : de-allocates a previously allocated real mode
memory}
procedure dosfree(selector : word);
begin
        { call the GO32 function with the selector }
        global_dos_free(selector);
end;

type
        VBEInfoBuf = packed record
                { contains 'VESA' if successful }
                Signature : array[0..3] of char;
                Version : Word;
                { pad to 512 bytes length }
                reserved : array[0..505] of byte;
        end;

var
        { selector to our real mode buffer }
        selector,
        { real mode segment address of buffer }
        segment : Word;

        { register structure to issue a software interrupt }
        r : trealregs;
        infobuf : VBEInfoBuf;

begin
        { first we reset the registers and infobuf variable }
```

```
        fillchar(r, sizeof(r), 0);
        fillchar(infobuf, sizeof(VBEInfoBuf), 0);
        { allocate real mode memory }
        dosalloc(selector, segment, sizeof(VBEInfoBuf));
        { check if an error occurred during allocation }
        if (int31error<>0) then begin
                Writeln('Error while allocating real mode memory, halting');
                halt;
        end;
        { request VBE 2.0 information, fill out information buffer }
        infobuf.Signature := 'VBE2';
        { copy buffer to the allocated real mode memory }
        dosmemput(segment, 0, infobuf, sizeof(infobuf));
        { issue the interrupt; remember : DI = 0 }
        r.ax := $4f00; r.es := segment;
        realintr($10, r);
        { copy buffer to our infobuf variable again }
        dosmemget(segment, 0, infobuf, sizeof(infobuf));
        { free allocated real mode memory, because we don't need it
        anymore }
        dosfree(selector);
        { check if interrupt call was successful }
        if (r.ax <> $4f) then begin
                { write message and exit, because the infobuf doesn't contain
                any useful data we could tell the user }
                Writeln('VBE BIOS extension not available, function call ',
                        'failed');
                halt;
        end;
        { check if buffer is valid }
        if (infobuf.signature[0] = 'V') and
                (infobuf.signature[1] = 'E') and
                (infobuf.signature[2] = 'S') and
                (infobuf.signature[3] = 'A') then begin
                Writeln('VBE version ', hi(infobuf.version), '.',
                        lo(infobuf.version), ' detected');
        end;
end.
```

54.19.36 global_dos_free

Synopsis: Free DOS memory block.

Declaration: `function global_dos_free(selector: Word) : Boolean`

Visibility: default

Description: Frees a previously allocated dos memory block.

Parameters:

selectorselector to the dos memory block.

Return value: `True` if successful, `False` otherwise.

Remark The descriptor allocated for the memory block is automatically freed and hence invalid for further use. This function should only be used for memory allocated by global_dos_alloc (833).

For an example, see global_dos_alloc (833).

Errors: Check the int31error (813) variable.

See also: global_dos_alloc (833)

54.19.37 inportb

Synopsis: Read byte from I/O port.

Declaration: `function inportb(port: Word) : Byte`

Visibility: default

Description: Reads 1 byte from the selected I/O port.

Parameters:

portthe I/O port number which is read.

Return values: Current I/O port value.

Errors: None.

See also: outportb (838), inportw (836), inportl (836)

54.19.38 inportl

Synopsis: Read longint from I/O port.

Declaration: `function inportl(port: Word) : LongInt`

Visibility: default

Description: Reads 1 longint from the selected I/O port.

Parameters:

portthe I/O port number which is read.

Return values: Current I/O port value.

Errors: None.

See also: outportb (838), inportb (836), inportw (836)

54.19.39 inportw

Synopsis: Read word from I/O port.

Declaration: `function inportw(port: Word) : Word`

Visibility: default

Description: Reads 1 word from the selected I/O port.

Parameters:

portthe I/O port number which is read.

Return values: Current I/O port value.

Errors: None.

See also: outportw (839), inportb (836), inportl (836)

54.19.40 lock_code

Synopsis: Lock code memory range.

Declaration: `function lock_code(functionaddr: pointer; size: LongInt) : Boolean`

Visibility: default

Description: Locks a memory range which is in the code segment selector.

Parameters:

functionaddraddress of the function to be locked.

sizesize in bytes to be locked.

Return values: `True` if successful, `False` otherwise.

For an example, see get_rm_callback (826).

Errors: Check the int31error (813) variable.

See also: lock_linear_region (837), lock_data (837), unlock_linear_region (849), unlock_data (848), unlock_code (848)

54.19.41 lock_data

Synopsis: Lock data memory range.

Declaration: `function lock_data(var data; size: LongInt) : Boolean`

Visibility: default

Description: Locks a memory range which resides in the data segment selector.

Parameters:

dataaddress of data to be locked.

sizelength of data to be locked.

Return values: `True` if successful, `False` otherwise.

For an example, see get_rm_callback (826).

Errors: Check the int31error (813) variable.

See also: lock_linear_region (837), lock_code (837), unlock_linear_region (849), unlock_data (848), unlock_code (848)

54.19.42 lock_linear_region

Synopsis: Lock linear memory region.

Declaration:
```
function lock_linear_region(linearaddr: LongInt; size: LongInt)
                                    : Boolean
```

Visibility: default

Description: Locks a memory region to prevent swapping of it.

Parameters:

linearaddrthe linear address of the memory are to be locked.

sizesize in bytes to be locked.

Return value: `True` if successful, False otherwise.

Errors: Check the int31error (813) variable.

See also: lock_data (837), lock_code (837), unlock_linear_region (849), unlock_data (848), unlock_code (848)

54.19.43 map_device_in_memory_block

Synopsis: Map a device into program's memory space.

Declaration:
```
function map_device_in_memory_block(handle: DWord; offset: DWord;
                                    pagecount: DWord; device: DWord)
                                     : Boolean
```

Visibility: default

Description: `map_device_in_memory_block` allows to map a device in memory. This function is a direct call of the extender. For more information about it's arguments, see the extender documentation.

54.19.44 outportb

Synopsis: Write byte to I/O port.

Declaration:
```
procedure outportb(port: Word; data: Byte)
```

Visibility: default

Description: Sends 1 byte of data to the specified I/O port.

Parameters:

portthe I/O port number to send data to.
datavalue sent to I/O port.

Return values: None.

Errors: None.

See also: inportb (836), outportl (839), outportw (839)

Listing: ./examples/go32ex/outport.pp

```
{ This example demonstrates the use of the outport functions.

It simply turns the PC's internal speaker on for 50 ms and off again
}
uses
        crt,
        go32;

begin
        { turn on speaker }
        outportb($61, $ff);
        { wait a little bit }
        delay(50);
        { turn it off again }
        outportb($61, $0);
end.
```

54.19.45 outportl

Synopsis: Write longint to I/O port.

Declaration: `procedure outportl(port: Word; data: LongInt)`

Visibility: default

Description: Sends 1 longint of data to the specified I/O port.

Parameters:

portthe I/O port number to send data to.
datavalue sent to I/O port.

Return values: None.

For an example, see outportb (838).

Errors: None.

See also: inportl (836), outportw (839), outportb (838)

54.19.46 outportw

Synopsis: Write word to I/O port.

Declaration: `procedure outportw(port: Word; data: Word)`

Visibility: default

Description: Sends 1 word of data to the specified I/O port.

Parameters:

portthe I/O port number to send data to.
datavalue sent to I/O port.

Return values: None.

For an example, see outportb (838).

Errors: None.

See also: inportw (836), outportl (839), outportb (838)

54.19.47 realintr

Synopsis: Simulate interrupt.

Declaration: `function realintr(intnr: Word; var regs: trealregs) : Boolean`

Visibility: default

Description: Simulates an interrupt in real mode.

Parameters:

intnrinterrupt number to issue in real mode.
regsregisters data structure.

Return values: The supplied registers data structure contains the values that were returned by the real mode interrupt. `True` if successful, `False` if not.

Remark The function transfers control to the address specified by the real mode interrupt vector of intnr. The real mode handler must return by executing an IRET.

Errors: Check the int31error (813) variable.

Listing: ./examples/go32ex/flags.pp

```
{ This example demonstrates the use of the flag constants in
conjunction with an interrupt call

In detail it checks if APM (advanced power management) is
available.

Int 15h 5300h - APM specification : Installation check
Input : AX = 5300h
        BX = device id of system BIOS (= 0000h)
Return : Carry clear if successful
        AH = major version (BCD)
        AL = minor version (BCD)
}

uses
        go32;

var
        r : trealregs;

begin
        { set register values and issue real mode interrupt call }
        r.ax := $5300;
        r.bx := 0;
        realintr($15, r);
        { check if carry clear and write a suited message }
        if ((r.flags and carryflag)=0) then begin
                Writeln('APM v', (r.ah and $f), '.',
                        (r.al shr 4), (r.al and $f), ' detected');
        end else
                Writeln('APM not present');
end.
```

54.19.48 request_linear_region

Synopsis: Request linear address region.

Declaration:
```
function request_linear_region(linearaddr: LongInt; size: LongInt;
                               var blockhandle: LongInt) : Boolean
```

Visibility: default

Description: `request_linear_region` requests a linear range of addresses of size `Size`, starting at `linearaddr`. If successful, `True` is returned, and a handle to the address region is returned in `blockhandle`.

Errors: On error, `False` is returned.

54.19.49 segment_to_descriptor

Synopsis: Map segment address to descriptor.

Declaration: `function segment_to_descriptor(seg: Word) : Word`

Visibility: default

Description: Maps a real mode segment (paragraph) address onto an descriptor that can be used by a protected mode program to access the same memory.

Parameters:

segthe real mode segment you want the descriptor to.

Return values: Descriptor to real mode segment address.

Remark The returned descriptors limit will be set to 64 kB. Multiple calls to this function with the same segment address will return the same selector. Descriptors created by this function can never be modified or freed. Programs which need to examine various real mode addresses using the same selector should use the function allocate_ldt_descriptors (813) and change the base address as necessary.

For an example, see seg_fillchar (841).

Errors: Check the int31error (813) variable.

See also: allocate_ldt_descriptors (813), free_ldt_descriptor (820), set_segment_base_address (846)

54.19.50 seg_fillchar

Synopsis: Fill segment with byte value.

Declaration: `procedure seg_fillchar(seg: Word; ofs: LongInt; count: LongInt; c: Char)`

Visibility: default

Description: Sets a memory area to a specific value.

Parameters:

segselector to memory area.

ofsoffset to memory.

countnumber of bytes to set.

cbyte data which is set.

Return values: None.

Notes: No range check is done in any way.

Errors: None.

See also: seg_move (843), seg_fillword (842), dosmemfillchar (809), dosmemfillword (809), dosmemget (809), dosmemput (810), dosmemmove (810)

Listing: ./examples/go32ex/vgasel.pp

```
{This example demonstrates the use of the segment_to_descriptor()
function.

It switches to VGA mode 13h (320x200x256 color), creates a selector
to the memory (based at $A000:0000), clears this memory with color
15 (white) and waits until the enter key is pressed }

uses go32;

var
        vgasel : Word;
        r : trealregs;

begin
        { set VGA mode 13h }
        r.eax := $13; realintr($10, r);
        { allocate descriptor to VGA memory quickly; it could be done
        with allocate_ldt_descriptors() too, but we would have to
        initialize it by ourselves... unlike segment_to_descriptor()
        which automatically sets the limit and the base address
        correctly }
        vgasel := segment_to_descriptor($A000);
        { simply fill the screen memory with color 15 }
        seg_fillchar(vgasel, 0, 64000, #15);
        { wait for a return press }
        readln;
        { back to text mode }
        r.eax := $3; realintr($10, r);
        { don't deallocate vgasel, that can't be done }
end.
```

54.19.51 seg_fillword

Synopsis: Fill segment with word value.

Declaration: `procedure seg_fillword(seg: Word; ofs: LongInt; count: LongInt; w: Word)`

Visibility: default

Description: Sets a memory area to a specific value.

Parameters:

segselector to memory area.

ofsoffset to memory.

countnumber of words to set.

wword data which is set.

Return values: None.

Notes: No range check is done in any way.

For an example, see allocate_ldt_descriptors (813).

Errors: None.

See also: seg_move (843), seg_fillchar (841), dosmemfillchar (809), dosmemfillword (809), dosmemget (809), dosmemput (810), dosmemmove (810)

54.19.52 seg_move

Synopsis: Move data between 2 locations.

Declaration:
```
procedure seg_move(sseg: Word; source: LongInt; dseg: Word;
                   dest: LongInt; count: LongInt)
```

Visibility: default

Description: Copies data between two memory locations.

Parameters:

ssegsource selector.

sourcesource offset.

dsegdestination selector.

destdestination offset.

countsize in bytes to copy.

Return values: None.

Remark Overlapping is only checked if the source selector is equal to the destination selector. No range check is done.

For an example, see allocate_ldt_descriptors (813).

Errors: None.

See also: seg_fillchar (841), seg_fillword (842), dosmemfillchar (809), dosmemfillword (809), dosmemget (809), dosmemput (810), dosmemmove (810)

54.19.53 set_descriptor_access_right

Synopsis: Set access rights to memory descriptor.

Declaration:
```
function set_descriptor_access_right(d: Word; w: Word) : Boolean
```

Visibility: default

Description: `set_descriptor_access_right` sets the access rights for descriptor `d` to `w`

54.19.54 set_exception_handler

Synopsis: Set exception handler.

Declaration:
```
function set_exception_handler(e: Byte; const intaddr: tseginfo)
                                        : Boolean
```

Visibility: default

Description: `set_exception_handler` sets the exception handler for exception `E` to `intaddr`. It returns `True` if the call was successful, `False` if not.

See also: get_exception_handler (822), set_pm_exception_handler (844)

54.19.55 set_page_attributes

Synopsis: ? No description available.

Declaration:
```
function set_page_attributes(handle: DWord; offset: DWord;
                             pagecount: DWord; buf: pointer) : Boolean
```

Visibility: default

54.19.56 set_pm_exception_handler

Synopsis: Set protected mode exception handler.

Declaration:
```
function set_pm_exception_handler(e: Byte; const intaddr: tseginfo)
                                  : Boolean
```

Visibility: default

Description: `set_pm_exception_handler` sets the protected mode exception handler for exception `E` to `intaddr`. It returns `True` if the call was successful, `False` if not.

See also: set_exception_handler (843), get_pm_exception_handler (825)

54.19.57 set_pm_interrupt

Synopsis: Set protected mode interrupt handler.

Declaration:
```
function set_pm_interrupt(vector: Byte; const intaddr: tseginfo)
                          : Boolean
```

Visibility: default

Description: Sets the address of the protected mode handler for an interrupt.

Parameters:

vectornumber of protected mode interrupt to set.

intaddrselector:offset address to the interrupt vector.

Return values: `True` if successful, `False` otherwise.

Remark The address supplied must be a valid `selector:offset` protected mode address.

Errors: Check the int31error (813) variable.

See also: get_pm_interrupt (825), set_rm_interrupt (846), get_rm_interrupt (830)

Listing: ./examples/go32ex/intpm.pp

```
{ This example shows how to redirect a software interrupt by
changing the protected mode handler of the DPMI host.

In more detail it hooks interrupt 1Ch which is called every
time the timer interrupt (int 08) is executed. This is the
preferred way to hook the timer, because int 1Ch is a software
interrupt which doesn't need so much initialization stuff
compared to hooking a hardware interrupt.
}
```

uses

```
        crt,
        go32;

const
        { interrupt number we want to hook }
        int1c = $1c;

var
        { 48 bit pointer to old interrupt handler }
        oldint1c : tseginfo;
        { 48 bit pointer to new interrupt handler }
        newint1c : tseginfo;

        { increased every time the interrupt is called  }
        int1c_counter : Longint;

        { the current data selector }
        int1c_ds : Word; external name '___v2prt0_ds_alias';

{ the actual handler code }
procedure int1c_handler; assembler;
asm
    cli
{ save all registers }
    pushw %ds
    pushw %ax
{ prepare segment registers for FPC procedure }
    movw %cs:int1c_ds, %ax
    movw %ax, %ds
{ simply increase the counter by one }
    incl int1c_counter
{ restore registers }
    popw %ax
    popw %ds
    sti
    iret
end;

var i : Longint;

begin
     { insert right handler data into new handler variable }
     newint1c.offset := @int1c_handler;
     newint1c.segment := get_cs;
     { get the old handler }
     get_pm_interrupt(int1c, oldint1c);
     WriteIn('-- Press any key to exit --');
     { set new handler }
     set_pm_interrupt(int1c, newint1c);
     { write the number of interrupts occurred }
     while (not keypressed) do begin
           gotoxy(1, wherey);
           write('Number of interrupts occurred : ', int1c_counter);
     end;
     { restore old handler }
     set_pm_interrupt(int1c, oldint1c);
end.
```

54.19.58 set_rm_interrupt

Synopsis: Set real mode interrupt handler.

Declaration:
```
function set_rm_interrupt(vector: Byte; const intaddr: tseginfo)
                                  : Boolean
```

Visibility: default

Description: Sets a real mode interrupt handler.

Parameters:

vectorthe interrupt vector number to set.

intaddraddress of new interrupt vector.

Return values: `True` if successful, otherwise `False`.

Remark The address supplied MUST be a real mode segment address, not a `selector:offset` address. So the interrupt handler must either reside in dos memory (below 1 Mb boundary) or the application must allocate a real mode callback address with get_rm_callback (826).

Errors: Check the int31error (813) variable.

See also: get_rm_interrupt (830), set_pm_interrupt (844), get_pm_interrupt (825), get_rm_callback (826)

54.19.59 set_segment_base_address

Synopsis: Set descriptor's base address.

Declaration:
```
function set_segment_base_address(d: Word; s: DWord) : Boolean
```

Visibility: default

Description: Sets the 32-bit linear base address of a descriptor.

Parameters:

dselector.

snew base address of the descriptor.

Errors: Check the int31error (813) variable.

See also: allocate_ldt_descriptors (813), get_segment_base_address (832), allocate_ldt_descriptors (813), set_segment_limit (846), get_segment_base_address (832), get_segment_limit (832)

54.19.60 set_segment_limit

Synopsis: Set descriptor limit.

Declaration:
```
function set_segment_limit(d: Word; s: DWord) : Boolean
```

Visibility: default

Description: Sets the limit of a descriptor.

Parameters:

dselector.

snew limit of the descriptor.

Return values: Returns `True` if successful, else `False`.

Remark The new limit specified must be the byte length of the segment - 1. Segment limits bigger than or equal to 1MB must be page aligned, they must have the lower 12 bits set.

For an example, see allocate_ldt_descriptors (813).

Errors: Check the int31error (813) variable.

See also: allocate_ldt_descriptors (813), set_segment_base_address (846), get_segment_limit (832), set_segment_limit (846)

54.19.61 tb_offset

Synopsis: Return DOS transfer buffer offset.

Declaration: `function tb_offset : LongInt`

Visibility: default

Description: `tb_offset` returns the DOS transfer buffer segment.

See also: transfer_buffer (848), tb_segment (847), tb_size (847)

54.19.62 tb_segment

Synopsis: Return DOS transfer buffer segment.

Declaration: `function tb_segment : LongInt`

Visibility: default

Description: `tb_segment` returns the DOS transfer buffer segment.

See also: transfer_buffer (848), tb_offset (847), tb_size (847)

54.19.63 tb_size

Synopsis: Return DOS transfer memory buffer size.

Declaration: `function tb_size : LongInt`

Visibility: default

Description: Returns the size of the pre-allocated dos memory buffer.

Return values: The size of the pre-allocated dos memory buffer. This block always seems to be 16k in size, but don't rely on this.

Errors: None.

See also: transfer_buffer (848), copyfromdos (817), copytodos (817)

54.19.64 transfer_buffer

Synopsis: Return offset of DOS transfer buffer.

Declaration: `function transfer_buffer : LongInt`

Visibility: default

Description: `transfer_buffer` returns the offset of the transfer buffer.

Errors: None.

See also: tb_size (847)

54.19.65 unlock_code

Synopsis: Unlock code segment.

Declaration: `function unlock_code(functionaddr: pointer; size: LongInt) : Boolean`

Visibility: default

Description: Unlocks a memory range which resides in the code segment selector.

Parameters:

functionaddraddress of function to be unlocked.

sizesize bytes to be unlocked.

Return value: `True` if successful, `False` otherwise.

For an example, see get_rm_callback (826).

Errors: Check the int31error (813) variable.

See also: unlock_linear_region (849), unlock_data (848), lock_linear_region (837), lock_data (837), lock_code (837)

54.19.66 unlock_data

Synopsis: Unlock data segment.

Declaration: `function unlock_data(var data; size: LongInt) : Boolean`

Visibility: default

Description: Unlocks a memory range which resides in the data segment selector.

Parameters:

dataaddress of memory to be unlocked.

sizesize bytes to be unlocked.

Return values: `True` if successful, `False` otherwise.

For an example, see get_rm_callback (826).

Errors: Check the int31error (813) variable.

See also: unlock_linear_region (849), unlock_code (848), lock_linear_region (837), lock_data (837), lock_code (837)

54.19.67 unlock_linear_region

Synopsis: Unlock linear memory region.

Declaration:
```
function unlock_linear_region(linearaddr: LongInt; size: LongInt)
                                    : Boolean
```

Visibility: default

Description: Unlocks a previously locked linear region range to allow it to be swapped out again if needed.

Parameters:

linearaddrlinear address of the memory to be unlocked.

sizesize bytes to be unlocked.

Return values: `True` if successful, `False` otherwise.

Errors: Check the int31error (813) variable.

See also: unlock_data (848), unlock_code (848), lock_linear_region (837), lock_data (837), lock_code (837)

54.20 tdpmiversioninfo

```
tdpmiversioninfo = record
  major : Byte;
  minor : Byte;
  flags
  : Word;
  cpu : Byte;
  master_pic : Byte;
  slave_pic : Byte;
end
```

`tdpmiversioninfo` describes the dpmi version information, as returned by get_dpmi_version (822). The CPU field can have the following values:

$02H 80286

$03H 80386

$04H 80486

$05H- Newer than 80486

The flags field is a bitmask with the following bits:

0 0 for 16 bit DPMI, 1 for 32-bit

1 0 for virtual 86 mode for reflected interrupts, 1 for return to real mode.

2 0 for no virtual memory support, 1 for virtual memory support.

54.21 tmeminfo

```
tmeminfo = record
  available_memory : LongInt;
  available_pages
  : LongInt;
  available_lockable_pages : LongInt;
  linear_space
  : LongInt;
  unlocked_pages : LongInt;
  available_physical_pages
  : LongInt;
  total_physical_pages : LongInt;
  free_linear_space
  : LongInt;
  max_pages_in_paging_file : LongInt;
  reserved0 : LongInt
  ;
  reserved1 : LongInt;
  reserved2 : LongInt;
end
```

`tmeminfo` Holds information about the memory allocation, etc.

NOTE: The value of a field is -1 (0ffffffffh) if the value is unknown, it's only guaranteed, that `available_memory` contains a valid value. The size of the pages can be determined by the get_page_size() function.

54.22 tseginfo

```
tseginfo = record
  offset : pointer;
  segment : Word;
end
```

This record is used to store a full 48-bit pointer. This may be either a protected mode selector:offset address or in real mode a segment:offset address, depending on application.

See also: Selectors and descriptors, dos memory access, Interrupt redirection

Chapter 55

Reference for unit 'gpm'

55.1 Used units

Table 55.1: Used units by unit 'gpm'

Name	Page
BaseUnix	146
System	1362

55.2 Overview

The GPM unit implements an interface to libgpm, the console program for mouse handling. This unit was created by Peter Vreman, and is only available on Linux.

When this unit is used, your program is linked to the C libraries, so you must take care of the C library version. Also, it will only work with version 1.17 or higher of the libgpm library.

55.3 Constants, types and variables

55.3.1 Constants

```
GPM_BOT = 2
```

Bottom of area.

```
GPM_B_LEFT = 4
```

Left mouse button identifier.

```
GPM_B_MIDDLE = 2
```

Middle mouse button identifier.

```
GPM_B_RIGHT = 1
```

Right mouse button identifier.

```
GPM_DOUBLE = 32
```

Mouse double click event.

```
GPM_DOWN = 4
```

Mouse button down event.

```
GPM_DRAG = 2
```

Mouse drag event.

```
GPM_ENTER = 512
```

Enter area event.

```
GPM_HARD = 256
```

?

```
GPM_LEAVE = 1024
```

Leave area event.

```
GPM_LFT = 4
```

Left side of area.

```
GPM_MAGIC = $47706D4C
```

Constant identifying GPM in Gpm_Open (858).

```
GPM_MFLAG = 128
```

Motion flag.

```
GPM_MOVE = 1
```

Mouse move event.

```
GPM_NODE_CTL = GPM_NODE_DEV
```

Control socket.

```
GPM_NODE_DEV = '/dev/gpmctl'
```

Device socket filename.

```
GPM_NODE_DIR = _PATH_VARRUN
```

Where to write socket.

```
GPM_NODE_DIR_MODE = 0775
```

Mode of socket.

```
GPM_NODE_FIFO = '/dev/gpmdata'
```

FIFO name.

```
GPM_NODE_PID = '/var/run/gpm.pid'
```

Name of PID file.

```
GPM_RGT = 8
```

Right side of area.

```
GPM_SINGLE = 16
```

Mouse single click event.

```
GPM_TOP = 1
```

Top of area.

```
GPM_TRIPLE = 64
```

Mouse triple click event.

```
GPM_UP = 8
```

Mouse button up event.

```
_PATH_DEV = '/dev/'
```

Location of /dev directory.

```
_PATH_VARRUN = '/var/run/'
```

Location of run PID files directory.

55.3.2 Types

```
Pgpmconnect = Pgpm_connect
```

Pointer to TGpmConnect (854) record.

```
Pgpmevent = Pgpm_event
```

Pointer to TGpmEvent (854) record.

```
Pgpmroi = Pgpm_roi
```

Pointer to TGpmRoi (854) record.

```
Pgpm_connect = ^TGpm_connect
```

Pointer to TGpm_Connect (861) record.

```
Pgpm_event = ^Tgpm_event
```

Pointer to TGpm_Event (861) record.

```
Pgpm_roi = ^Tgpm_roi
```

Pointer to Tgpm_roi (862) record.

```
Tgpmconnect = Tgpm_connect
```

Alias for TGpm_Connect (861) record.

```
TGpmEtype = LongInt
```

Type for event type.

```
Tgpmevent = Tgpm_event
```

Alias for TGPM_EVent (861) record.

```
TGpmHandler = function(var event: Tgpmevent; clientdata: pointer)
  : LongInt
```

Mouse event handler callback.

```
TGpmMargin = LongInt
```

Type to hold area margin.

```
Tgpmroi = Tgpm_roi
```

Alias for TGpm_roi (862)Record.

55.3.3 Variables

```
gpm_current_roi : Pgpm_roi
```

Internal gpm library variable. Do not use.

```
gpm_handler : TGpmHandler
```

Internal gpm library variable. Do not use.

`gpm_roi : Pgpm_roi`

Internal gpm library variable. Do not use.

`gpm_roi_data : pointer`

Internal gpm library variable. Do not use.

`gpm_roi_handler : TGpmHandler`

Internal gpm library variable. Do not use.

55.4 Procedures and functions

55.4.1 Gpm_AnyDouble

Synopsis: Check whether event has double click event.

Declaration: `function Gpm_AnyDouble(EventType: LongInt) : Boolean`

Visibility: default

Description: `Gpm_AnyDouble` returns `True` if `EventType` contains the `GPM_DOUBLE` flag, `False` otherwise.

Errors: None.

See also: Gpm_StrictSingle (860), Gpm_AnySingle (855), Gpm_StrictDouble (860), Gpm_StrictTriple (860), Gpm_AnyTriple (855)

55.4.2 Gpm_AnySingle

Synopsis: Check whether event has a single click event.

Declaration: `function Gpm_AnySingle(EventType: LongInt) : Boolean`

Visibility: default

Description: `Gpm_AnySingle` returns `True` if `EventType` contains the `GPM_SINGLE` flag, `False` otherwise.

Errors: None.

See also: Gpm_StrictSingle (860), Gpm_AnyDouble (855), Gpm_StrictDouble (860), Gpm_StrictTriple (860), Gpm_AnyTriple (855)

55.4.3 Gpm_AnyTriple

Synopsis: Check whether event has a triple click event.

Declaration: `function Gpm_AnyTriple(EventType: LongInt) : Boolean`

Visibility: default

Description: `Gpm_AnySingle` returns `True` if `EventType` contains the `GPM_TRIPLE` flag, `False` otherwise.

Errors: None.

See also: Gpm_StrictSingle (860), Gpm_AnyDouble (855), Gpm_StrictDouble (860), Gpm_StrictTriple (860), Gpm_AnySingle (855)

55.4.4 gpm_close

Synopsis: Close connection to GPM server.

Declaration: `function gpm_close : LongInt`

Visibility: default

Description: `Gpm_Close` closes the current connection, and pops the connection stack; this means that the previous connection becomes active again.

The function returns -1 if the current connection is not the last one, and it returns 0 if the current connection is the last one.

for an example, see Gpm_GetEvent (857).

Errors: None.

See also: Gpm_Open (858)

55.4.5 gpm_fitvalues

Synopsis: Change coordinates to fit physical screen.

Declaration: `function gpm_fitvalues(var x: LongInt; var y: LongInt) : LongInt`

Visibility: default

Description: `Gpm_fitValues` changes `x` and `y` so they fit in the visible screen. The actual mouse pointer is not affected by this function.

Errors: None.

See also: Gpm_FitValuesM (856)

55.4.6 gpm_fitvaluesM

Synopsis: Change coordinates to fit margin.

Declaration:
```
function gpm_fitvaluesM(var x: LongInt; var y: LongInt; margin: LongInt)
                        : LongInt
```

Visibility: default

Description: `Gpm_FitValuesM` changes `x` and `y` so they fit in the margin indicated by `margin`. If `margin` is -1, then the values are fitted to the screen. The actual mouse pointer is not affected by this function.

Errors: None.

See also: Gpm_FitValues (856)

55.4.7 gpm_getevent

Synopsis: Get event from event queue.

Declaration: `function gpm_getevent(var event: Tgpm_event) : LongInt`

Visibility: default

Description: `Gpm_GetEvent` Reads an event from the file descriptor `gpm_fd`. This file is only for internal use and should never be called by a client application.

It returns 1 on success, and -1 on failure.

Errors: On error, -1 is returned.

See also: Gpm_GetSnapshot (858)

Listing: ./examples/gpmex/gpmex.pp

```
program gpmex;

{
  Example program to demonstrate the use of the gpm unit.
}

uses gpm;

var
  connect : TGPMConnect;
  event : tgpmevent;

begin
  connect.EventMask:=GPM_MOVE or GPM_DRAG or GPM_DOWN or GPM_UP;
  connect.DefaultMask:=0;
  connect.MinMod:=0;
  connect.MaxMod:=0;
  if Gpm_Open(connect,0)=-1 then
    begin
    Writeln('No mouse handler present.');
    Halt(1);
    end;
  Writeln('Click right button to end.');
  Repeat
    gpm_getevent(Event);
    With Event do
      begin
        Write('Pos = (',X,',',Y,') Buttons : (');
        if (buttons and Gpm_b_left)<>0 then
          write('left ');
        if (buttons and Gpm_b_right)<>0 then
          write('right ');
        if (buttons and Gpm_b_middle)<>0 then
          Write('middle ');
        Write(') Event : ');
        Case EventType and $F of
          GPM_MOVE: write('Move');
          GPM_DRAG: write('Drag');
          GPM_DOWN: write('Down');
          GPM_UP: write('Up');
        end;
```

```
        WriteLn;
      end;
  Until (Event.Buttons and gpm_b_right)<>0;
  gpm_close;
end.
```

55.4.8 gpm_getsnapshot

Synopsis: Return servers' current image of mouse state.

Declaration:
```
function gpm_getsnapshot(eptr: Pgpmevent) : LongInt
function gpm_getsnapshot(var eptr: Tgpmevent) : LongInt
```

Visibility: default

Description: `Gpm_GetSnapshot` returns the picture that the server has of the current situation in `Event`. This call will not read the current situation from the mouse file descriptor, but returns a buffered version.

The function returns the number of mouse buttons, or -1 if this information is not available.

Errors: None.

See also: Gpm_GetEvent (857)

55.4.9 gpm_lowerroi

Synopsis: Lower a region of interest in the stack.

Declaration:
```
function gpm_lowerroi(which: Pgpm_roi; after: Pgpm_roi) : Pgpm_roi
```

Visibility: default

Description: `Gpm_LowerRoi` lowers the region of interest `which` after `after`. If `after` is `Nil`, the region of interest is moved to the bottom of the stack.

The return value is the new top of the region-of-interest stack.

Errors: None.

See also: Gpm_RaiseRoi (859), Gpm_PopRoi (859), Gpm_PushRoi (859)

55.4.10 gpm_open

Synopsis: Open connection to GPM server.

Declaration:
```
function gpm_open(var conn: Tgpm_connect; flag: LongInt) : LongInt
```

Visibility: default

Description: `Gpm_Open` opens a new connection to the mouse server. The connection is described by the fields of the `conn` record of type TGPMConnect (854).

if `Flag` is 0, then the application only receives events that come from its own terminal device. If it is negative it will receive all events. If the value is positive then it is considered a console number to which to connect.

The return value is -1 on error, or the file descriptor used to communicate with the client. Under an X-Term the return value is -2.

for an example, see Gpm_GetEvent (857).

Errors: On Error, the return value is -1.

See also: Gpm_Open (858)

55.4.11 gpm_poproi

Synopsis: Pop region of interest from the stack.

Declaration:
```
function gpm_poproi(which: Pgpm_roi) : Pgpm_roi
```

Visibility: default

Description: `Gpm_PopRoi` pops the topmost region of interest from the stack. It returns the next element on the stack, or `Nil` if the current element was the last one.

Errors: None.

See also: Gpm_RaiseRoi (859), Gpm_LowerRoi (858), Gpm_PushRoi (859)

55.4.12 gpm_pushroi

Synopsis: Push region of interest on the stack.

Declaration:
```
function gpm_pushroi(x1: LongInt; y1: LongInt; x2: LongInt;
                     y2: LongInt; mask: LongInt; fun: TGpmHandler;
                     xtradata: pointer) : Pgpm_roi
```

Visibility: default

Description: `Gpm_PushRoi` puts a new *region of interest* on the stack. The region of interest is defined by a rectangle described by the corners `(X1,Y1)` and `(X2,Y2)`.

The `mask` describes which events the handler {fun} will handle; `ExtraData` will be put in the `xtradata` field of the {TGPM_Roi} record passed to the `fun` handler.

Errors: None.

See also: Gpm_RaiseRoi (859), Gpm_PopRoi (859), Gpm_LowerRoi (858)

55.4.13 gpm_raiseroi

Synopsis: Raise region of interest in the stack.

Declaration:
```
function gpm_raiseroi(which: Pgpm_roi; before: Pgpm_roi) : Pgpm_roi
```

Visibility: default

Description: `Gpm_RaiseRoi` raises the *region of interest*`which` till it is on top of region `before`. If `before` is nil then the region is put on top of the stack. The returned value is the top of the stack.

Errors: None.

See also: Gpm_PushRoi (859), Gpm_PopRoi (859), Gpm_LowerRoi (858)

55.4.14 gpm_repeat

Synopsis: Check for presence of mouse event.

Declaration: `function gpm_repeat(millisec: LongInt) : LongInt`

Visibility: default

Description: `Gpm_Repeat` returns 1 of no mouse event arrives in the next `millisec` miiliseconds, it returns 0 otherwise.

Errors: None.

See also: Gpm_GetEvent (857)

55.4.15 Gpm_StrictDouble

Synopsis: Check whether event contains only a double-click event.

Declaration: `function Gpm_StrictDouble(EventType: LongInt) : Boolean`

Visibility: default

Description: `Gpm_StrictDouble` returns true if `EventType` contains only a doubleclick event, `False` otherwise.

Errors: None.

See also: Gpm_StrictSingle (860), Gpm_AnyTriple (855), Gpm_AnyDouble (855), Gpm_StrictTriple (860), Gpm_AnySingle (855)

55.4.16 Gpm_StrictSingle

Synopsis: Check whether event contains only a single-click event.

Declaration: `function Gpm_StrictSingle(EventType: LongInt) : Boolean`

Visibility: default

Description: `Gpm_StrictDouble` returns `True` if `EventType` contains only a singleclick event, `False` otherwise.

Errors: None.

See also: Gpm_AnyTriple (855), Gpm_StrictDouble (860), Gpm_AnyDouble (855), Gpm_StrictTriple (860), Gpm_AnySingle (855)

55.4.17 Gpm_StrictTriple

Synopsis: Check whether event contains only a triple-click event.

Declaration: `function Gpm_StrictTriple(EventType: LongInt) : Boolean`

Visibility: default

Description: `Gpm_StrictTriple` returns true if `EventType` contains only a triple click event, `False` otherwise.

Errors: None.

See also: Gpm_AnyTriple (855), Gpm_StrictDouble (860), Gpm_AnyDouble (855), Gpm_StrictSingle (860), Gpm_AnySingle (855)

55.5 Tgpm_connect

```
Tgpm_connect = record
  eventMask : Word;
  defaultMask : Word;
  minMod : Word;
  maxMod : Word;
  pid : LongInt;
  vc : LongInt;
  end
```

GPM server connection information.

55.6 Tgpm_event

```
Tgpm_event = record
  buttons : Byte;
  modifiers : Byte;
  vc : Word
  ;
  dx : Word;
  dy : Word;
  x : Word;
  y : Word;
  EventType :
  TGpmEtype;
  clicks : LongInt;
  margin : TGpmMargin;
  wdx : Word
  ;
  wdy : Word;
end
```

`Tgpm_event` describes the events that are reported by GPM.

55.7 Tgpm_roi

```
Tgpm_roi = record
  xmin : Integer;
  xmax : Integer;
  ymin : Integer
  ;
  ymax : Integer;
  minmod : Word;
  maxmod : Word;
  eventmask
  : Word;
  owned : Word;
  handler : TGpmHandler;
  clientdata : pointer
  ;
```

```
  prev : Pgpm_roi;
  next : Pgpm_roi;
end
```

Record used to define regions of interest.

Chapter 56

Reference for unit 'Graph'

56.1 Used units

Table 56.1: Used units by unit 'Graph'

Name	Page
System	1362

56.2 Overview

This document describes the GRAPH unit for Free Pascal, for all platforms. The unit was first written for dos by Florian Klaempfl, but was later completely rewritten by Carl-Eric Codere to be completely portable. The unit is provided for compatibility only: It is recommended to use more modern graphical systems. The graph unit will allow to recompile old programs. They will work to some extent, but if the application has heavy graphical needs, it's recommended to use another set of graphical routines, suited to the platform the program should work on.

56.3 Categorized functions: Text and font handling.

Functions to set texts on the screen.

Table 56.2:

Name	Description
GetTextSettings (902)	Get current text settings
InstallUserFont (905)	Install a new font
OutText (906)	Write text at current cursor position
OutTextXY (894)	Write text at coordinates X,Y
RegisterBGIFont (908)	Register a new font
SetTextJustify (911)	Set text justification
SetTextStyle (911)	Set text style
SetUserCharSize (912)	Set text size
TextHeight (913)	Calculate height of text
TextWidth (913)	Calculate width of text

56.4 Categorized functions: Filled drawings.

Functions for drawing filled regions.

Table 56.3:

Name	Description
Bar3D (895)	Draw a filled 3D-style bar
Bar (895)	Draw a filled rectangle
FloodFill (897)	Fill starting from coordinate
FillEllipse (897)	Draw a filled ellipse
FillPoly (897)	Draw a filled polygon
GetFillPattern (899)	Get current fill pattern
GetFillSettings (899)	Get current fill settings
SetFillPattern (909)	Set current fill pattern
SetFillStyle (909)	Set current fill settings

56.5 Categorized functions: Drawing primitives.

Functions for simple drawing.

Table 56.4:

Name	Description
Arc (895)	Draw an arc
Circle (892)	Draw a complete circle
DrawPoly (896)	Draw a polygon with N points
Ellipse (897)	Draw an ellipse
GetArcCoords (898)	Get arc coordinates
GetLineSettings (900)	Get current line drawing settings
Line (894)	Draw line between 2 points
LineRel (905)	Draw line relative to current position
LineTo (906)	Draw line from current position to absolute position
MoveRel (906)	Move cursor relative to current position
MoveTo (906)	Move cursor to absolute position
PieSlice (907)	Draw a pie slice
PutPixel (894)	Draw 1 pixel
Rectangle (907)	Draw a non-filled rectangle
Sector (908)	Draw a sector
SetLineStyle (910)	Set current line drawing style

56.6 Categorized functions: Color management.

All functions related to color management.

Table 56.5:

Name	Description
GetBkColor (893)	Get current background color
GetColor (898)	Get current foreground color
GetDefaultPalette (898)	Get default palette entries
GetMaxColor (900)	Get maximum valid color
GetPaletteSize (902)	Get size of palette for current mode
GetPixel (893)	Get color of selected pixel
GetPalette (902)	Get palette entry
SetAllPalette (894)	Set all colors in palette
SetBkColor (894)	Set background color
SetColor (909)	Set foreground color
SetPalette (911)	Set palette entry
SetRGBPalette (895)	Set palette entry with RGB values

56.7 Categorized functions: Screen management.

General drawing screen management functions.

Table 56.6:

Name	Description
ClearViewPort (893)	Clear the current viewport
GetImage (893)	Copy image from screen to memory
GetMaxX (901)	Get maximum X coordinate
GetMaxY (901)	Get maximum Y coordinate
GetX (903)	Get current X position
GetY (903)	Get current Y position
ImageSize (893)	Get size of selected image
GetViewSettings (902)	Get current viewport settings
PutImage (894)	Copy image from memory to screen
SetActivePage (894)	Set active video page
SetAspectRatio (908)	Set aspect ratio for drawing routines
SetViewPort (912)	Set current viewport
SetVisualPage (895)	Set visual page
SetWriteMode (913)	Set write mode for screen operations

56.8 Categorized functions: Initialization.

Initialization of the graphics screen.

Table 56.7:

Name	Description
ClearDevice (896)	Empty the graphics screen
CloseGraph (896)	Finish drawing session, return to text mode
DetectGraph (896)	Detect graphical modes
GetAspectRatio (898)	Get aspect ratio of screen
GetModeRange (901)	Get range of valid modes for current driver
GraphDefaults (903)	Set defaults
GetDriverName (899)	Return name of graphical driver
GetGraphMode (900)	Return current or last used graphics mode
GetMaxMode (900)	Get maximum mode for current driver
GetModeName (901)	Get name of current mode
GraphErrorMsg (903)	String representation of graphical error
GraphResult (904)	Result of last drawing operation
InitGraph (904)	Initialize graphics drivers
InstallUserDriver (905)	Install a new driver
RegisterBGIDriver (907)	Register a new driver
RestoreCRTMode (908)	Go back to text mode
SetGraphMode (910)	Set graphical mode

56.9 Target specific issues: Linux.

There are several issues on Linux that need to be taken care of:

The Linux version of the Graph unit uses the libvga library. This library works on the console, not under X.

If you get an error similar to

```
/usr/bin/ld: cannot find -lvga
```

This can mean one of two things: either libvga and it's development package is not installed properly, or the directory where it is installed is not in the linker path.

To remedy the former, you should install both the libvga package and libvga-devel package (or compile and install from scratch).

To remedy the latter, you should add the path to the compiler command-line using the `-Fl` option.

Programs using libvga need root privileges to run. You can make them setuid root with the following command:

```
chown root.root myprogram
chmod u+s myprogram
```

The libvga library will give up the root privileges after it is initialized.

there is an experimental version of the Graphics library available that uses GGI to do all the drawing, but it is not well tested. It's called ggigraph and is distributed in source form only.

Do not use the CRT unit together with the Graph unit: the console may end up in an unusable state. Instead, the ncurses unit may function fine.

56.10 Target specific issues: DOS.

VESA modes (i.e., anything but 320x200x256 and 640x480x16) do not work under most installations of Windows NT, Windows 2000 and Windows XP. They also do not work for some people under Windows 98 and Windows ME, depending on their graphics drivers. However, the graph unit cannot detect this, because no errors are returned from the system. In such cases, the screen simply turns black, or will show garbage.

Nothing can be done about this, the reason is missing or buggy support in the graphics drivers of the operating system.

56.11 A word about mode selection.

The graph unit was implemented for compatibility with the old Turbo Pascal graph unit. For this reason, the mode constants as they were defined in the Turbo Pascal graph unit are retained.

However, since

1. Video cards have evolved very much
2. Free Pascal runs on multiple platforms

it was decided to implement new mode and graphic driver constants, which are more independent of the specific platform the program runs on.

In this section we give a short explanation of the new mode system. the following drivers were defined:

```
D1bit = 11;
D2bit = 12;
```

```
D4bit = 13;
D6bit = 14;  { 64 colors Half-brite mode - Amiga }
D8bit = 15;
D12bit = 16; { 4096 color modes HAM mode - Amiga }
D15bit = 17;
D16bit = 18;
D24bit = 19; { not yet supported }
D32bit = 20; { not yet supported }
D64bit = 21; { not yet supported }

lowNewDriver = 11;
highNewDriver = 21;
```

Each of these drivers specifies a desired color-depth.

The following modes have been defined:

```
detectMode = 30000;
m320x200 = 30001;
m320x256 = 30002; { amiga resolution (PAL) }
m320x400 = 30003; { amiga/atari resolution }
m512x384 = 30004; { mac resolution }
m640x200 = 30005; { vga resolution }
m640x256 = 30006; { amiga resolution (PAL) }
m640x350 = 30007; { vga resolution }
m640x400 = 30008;
m640x480 = 30009;
m800x600 = 30010;
m832x624 = 30011; { mac resolution }
m1024x768 = 30012;
m1280x1024 = 30013;
m1600x1200 = 30014;
m2048x1536 = 30015;

lowNewMode = 30001;
highNewMode = 30015;
```

These modes start at 30000 because Borland specified that the mode number should be ascending with increasing X resolution, and the new constants shouldn't interfere with the old ones.

The above constants can be used to set a certain color depth and resolution, as demonstrated in the below example.

If other modes than the ones above are supported by the graphics card, you will not be able to select them with this mechanism.

For this reason, there is also a 'dynamic' mode number, which is assigned at run-time. This number increases with increasing X resolution. It can be queried with the `getmoderange` call. This call will return the range of modes which are valid for a certain graphics driver. The numbers are guaranteed to be consecutive, and can be used to search for a certain resolution, as in the second example below.

Thus, the `getmoderange` function can be used to detect all available modes and drivers, as in the third example below:

Listing: ./examples/graphex/inigraph1.pp

```
Program inigraph1;
```

```
{ Program to demonstrate static graphics mode selection }

uses graph;


const
  TheLine = 'We are now in 640 x 480 x 256 colors!'+
            ' (press <Return> to continue)';

var
  gd, gm, lo, hi, error,tw,th: integer;
  found: boolean;

begin
  { We want an 8 bit mode }
  gd := D8bit;
  gm := m640x480;
  initgraph(gd,gm,'');
  { Make sure you always check graphresult! }
  error := graphResult;
  if (error <> grOk) Then
    begin
    writeln('640x480x256 is not supported!');
    halt(1)
    end;
  { We are now in 640x480x256 }
  setColor(cyan);
  rectangle(0,0,getmaxx,getmaxy);
  { Write a nice message in the center of the screen }
  setTextStyle(defaultFont,horizDir,1);
  tw:=TextWidth(TheLine);
  th:=TextHeight(TheLine);
  outTextXY((getMaxX - TW) div 2,
            (getMaxY - TH) div 2,TheLine);
  { Wait for return }
  readln;
  { Back to text mode }
  closegraph;
end.
```

Listing: ./examples/graphex/inigraph2.pp

```
Program inigraph2;

{ Program to demonstrate dynamic graphics mode selection }

uses graph;

const
  TheLine = 'We are now in 640 x 480 x 256 colors!'+
            ' (press <Return> to continue)';

var
  th,tw,gd, gm, lo, hi, error: integer;
  found: boolean;

begin
  { We want an 8 bit mode }
```

```
  gd := D8bit;
  { Get all available resolutions for this bitdepth }
  getmoderange(gd,lo,hi);
  { If the highest available mode number is -1,
    no resolutions are supported for this bitdepth  }
  if hi = -1 then
    begin
    writeln('no 8 bit modes supported!');
    halt
    end;
  found := false;
  { Search all resolutions for 640x480 }
  for gm := lo to hi do
    begin
    initgraph(gd,gm,'');
    { Make sure you always check graphresult! }
    error := graphResult;
    if (error = grOk) and
       (getmaxx = 639) and (getmaxy = 479) then
      begin
      found := true;
      break;
      end;
    end;
  if not found then
      CloseGraph();
    begin
    writeln('640x480x256 is not supported!');
    halt(1)
    end;
  { We are now in 640x480x256 }
  setColor(cyan);
  rectangle(0,0,getmaxx,getmaxy);
  { Write a nice message in the center of the screen }
  setTextStyle(defaultFont,horizDir,1);
  TW:=TextWidth(TheLine);
  TH:=TextHeight(TheLine);
  outTextXY((getMaxX - TW) div 2,
            (getMaxY - TH) div 2,TheLine);
  { Wait for return }
  readln;
  { Back to text mode }
  closegraph;
end.
```

Listing: ./examples/graphex/modrange.pp

```
Program GetModeRange_Example;

{ This program demonstrates how to find all available graph modes }

uses graph;


const
  { Currently, only 4, 8, 15 and 16 bit modes are supported
    but this may change in the future }
  gdnames: array[D4bit..D16bit] of string[6] =
    ('4 bit','6 bit','8 bit','12 bit','15 bit','16 bit');
```

```
procedure WriteRes(const depth : integer);
var
  tw, th : integer;
  v, text : String;
begin
  text := 'Current resolution is '; str(getmaxx+1, v);
  text := text + v + 'x'; str(getmaxy+1, v);
  text := text + v + 'x' + gdnames[depth];
  setTextStyle(defaultFont,horizDir,1);
  TW:=TextWidth(text);
  TH:=TextHeight(text);
  outTextXY((getMaxX - TW) div 2,
            (getMaxY - TH) div 2,text);
end;

var
  t: text;
  line : string;
  gd, c, low, high, res: integer;
begin
  assign(t,'modes.txt');
  rewrite(t);
  close(t);
  for gd := D4bit to D16bit do
    begin
    { Get the available mode numbers for this driver }
    getModeRange(gd,low,high);
    append(t);
    write(t,gdnames[gd]);
    Writeln(t,': low modenr = ',low,', high modenr = ',high);
    close(t);
    { If high is -1,
       no resolutions are supported for this bitdepth }
    if high = -1 then
      begin
      append(t);
      writeln(t,'  No modes supported!');
      writeln(t);
      close(t);
      end
    else
      { Enter all supported resolutions for this bitdepth
        and write their characteristics to the file }
      for c := low to high do
        begin
        append(t);
        writeln(t,'  testing mode nr ',c);
        close(t);
        initgraph(gd,c,'');
        res := graphresult;
        append(t);
        { An error occurred when entering the mode? }
        if res <> grok then
          writeln(t,grapherrormsg(res))
        else
          begin
          write(t,'maxx: ',getmaxx,', maxy: ',getmaxy);
```

```
          WriteLn(t,', maxcolor: ',getmaxcolor);
          closegraph;
          end;
        writeln(t);
          WriteRes(gd);
        close(t);
        end;
    append(t);
    writeln(t);
    close(t);
    end;
  WriteLn('All supported modes are listed in modes.txt files');
end.
```

56.12 Requirements.

The unit Graph exports functions and procedures for graphical output. It requires at least a VGA-compatible Card or a VGA-Card with software-driver (min. **512Kb** video memory).

56.13 Constants, types and variables

56.13.1 Constants

```
AndPut = 3
```

Draw operation: use AND.

```
AnsiToASCIITransTable : TCharsetTransTable = (#$00, #$01, #$02, #
  $03, #$04, #$05, #$06, #$07, #$08, #$09, #$0a, #$0b, #$0c, #$0d,
  #$0e, #$0f, #$10, #$11, #$12, #$13, #$14, #$15, #$16, #$17, #$18,
  #$19, #$1a, #$1b, #$1c, #$1d, #$1e, #$1f, #$20, #$21, #$22, #$23
  , #$24, #$25, #$26, #$27, #$28, #$29, #$2a, #$2b, #$2c, #$2d, #$2e
  , #$2f, #$30, #$31, #$32, #$33, #$34, #$35, #$36, #$37, #$38, #$39
  , #$3a, #$3b, #$3c, #$3d, #$3e, #$3f, #$40, #$41, #$42, #$43, #$44
  , #$45, #$46, #$47, #$48, #$49, #$4a, #$4b, #$4c, #$4d, #$4e, #$4f
  , #$50, #$51, #$52, #$53, #$54, #$55, #$56, #$57, #$58, #$59, #$5a
  , #$5b, #$5c, #$5d, #$5e, #$5f, #$60, #$61, #$62, #$63, #$64, #$65
  , #$66, #$67, #$68, #$69, #$6a, #$6b, #$6c, #$6d, #$6e, #$6f, #$70
  , #$71, #$72, #$73, #$74, #$75, #$76, #$77, #$78, #$79, #$7a, #$7b
  , #$7c, #$7d, #$7e, #$7f, '?', '?', '?', '?', '?', '?', '?', '?',
  '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?',
  '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', #$ff, #$ad
  , #$9b, #$9c, '?', #$9d, '?', '?', '?', '?', #$a6, #$ae, #$aa, '?'
  , '?', '?', #$f8, #$f1, #$fd, '?', '?', #$e6, '?', #$fa, '?', '?'
  , #$a7, #$af, #$ac, #$ab, '?', #$a8, '?', '?', '?', '?', #$8e, #$8f
  , #$92, #$80, '?', #$90, '?', '?', '?', '?', '?', '?', '?', #$a5,
  '?', '?', '?', '?', #$99, '?', '?', '?', '?', '?', #$9a, '?', '?'
  , #$e1, #$85, #$a0, #$83, '?', #$84, #$86, #$91, #$87, #$8a, #$82
  , #$88, #$89, #$8d, #$a1, #$8c, #$8b, '?', #$a4, #$95, #$a2, #$93
  , '?', #$94, #$f6, '?', #$97, #$a3, #$96, #$81, '?', '?', #$98)
```

Default ansi transliteration table.

```
BkSlashFill = 5
```

Fill style: Diagonal (backslash) lines.

```
black = 0
```

Color code: black.

```
blue = 1
```

Color code: blue.

```
BoldFont = 10
```

Font number: Bold font.

```
BottomText = 0
```

Vertical text alignment: Align text to bottom.

```
brown = 6
```

Color code: brown.

```
CenterLn = 2
```

Line style: centered line.

```
CenterText = 1
```

Horizontal text alignment: Center text.

```
CGA = 1
```

Graphic driver for CGA cards.

```
CGAC0 = 0
```

CGA Graphic driver mode C0.

```
CGAC1 = 1
```

CGA Graphic driver mode C1.

```
CGAC2 = 2
```

CGA Graphic driver mode C2.

```
CGAC3 = 3
```

CGA Graphic driver mode C3.

```
CGAHi = 4
```

CGA Graphic driver Hi-res mode.

```
ClipOff = False
```

Viewport clipping off.

```
ClipOn = True
```

Viewport clipping on.

```
CloseDotFill = 11
```

Fill style: Closely spaced dotted lines.

```
CopyPut = 0
```

Draw operation: use Copy.

```
CurrentDriver = - 128
```

Currently used driver.

```
cyan = 3
```

Color code: Cyan.

```
D12bit = 16
```

Mode: Depth 12 bit.

```
D15bit = 17
```

Mode: Depth 15 bit.

```
D16bit = 18
```

Mode: Depth 16 bit.

```
D1bit = 11
```

Mode: Depth 1 bit.

```
D24bit = 19
```

Mode: Depth 24 bit.

```
D2bit = 12
```

Mode: Depth 2 bit.

```
D32bit = 20
```

Mode: Depth 32 bit.

```
D4bit = 13
```

Mode: Depth 4 bit.

```
D64bit = 21
```

Mode: Depth 64 bit.

```
D6bit = 14
```

Mode: Depth 6 bit.

```
D8bit = 15
```

Mode: Depth 8 bit.

```
darkgray = 8
```

Color code: Dark gray.

```
DashedLn = 3
```

Line style: dashed line.

```
Default = 0
```

Default mode.

```
DefaultFont = 0
```

Font number: Normal font.

```
Detect = 0
```

Mode: Detect mode.

```
detectMode = 30000
```

Mode: Autodetect optimal mode.

```
DottedLn = 1
```

Line style: Dotted line.

```
DrawTextBackground : Boolean = False
```

Should the background of texts be drawn or should it be left untouched ?

```
EGA = 3
```

Graphic driver for EGA cards.

```
EGA64 = 4
```

Graphic driver for EGA 64 cards.

```
EGA64Hi = 1
```

EGA64 graphic driver high resolution mode.

```
EGA64Lo = 0
```

EGA64 graphic driver low resolution mode.

```
EGABlack = 0
```

Color code: EGA Black.

```
EGABlue = 1
```

Color code: EGA blue.

```
EGABrown = 20
```

Color code: EGA brown.

```
EGACyan = 3
```

Color code: EGA cyan.

```
EGADarkgray = 56
```

Color code: EGA dark gray.

```
EGAGreen = 2
```

Color code: EGA green.

```
EGAHi = 1
```

EGA graphic driver high resolution mode.

```
EGALightblue = 57
```

Color code: EGA Light blue.

```
EGALightcyan = 59
```

Color code: EGA Light cyan.

```
EGALightgray = 7
```

Color code: EGA Light gray.

```
EGALightgreen = 58
```

Color code: EGA Light green.

```
EGALightmagenta = 61
```

Color code: EGA light magenta.

```
EGALightred = 60
```

Color code: EGA light red.

```
EGALo = 0
```

EGA graphic driver low resolution mode.

```
EGAMagenta = 5
```

Color code: EGA magenta.

```
EGAMono = 5
```

Graphic driver for EGA monochrome cards.

```
EGAMonoHi = 3
```

EGAMono graphic driver high resolution mode.

```
EGARed = 4
```

Color code: EGA red.

```
EGAWhite = 63
```

Color code: EGA white.

```
EGAYellow = 62
```

Color code: EGA yellow.

```
EmptyFill = 0
```

Fill style: Do not fill.

```
EuroFont = 9
```

Font number: ?

```
fillpatternTable : Array[0..12] of FillPatternType = (($00, $00,
  $00, $00, $00, $00, $00, $00), ($ff, $ff, $ff, $ff, $ff, $ff, $ff
  , $ff), ($ff, $ff, $00, $00, $ff, $ff, $00, $00), ($01, $02, $04,
  $08, $10, $20, $40, $80), ($07, $0e, $1c, $38, $70, $e0, $c1, $83
  ), ($07, $83, $c1, $e0, $70, $38, $1c, $0e), ($5a, $2d, $96, $4b,
  $a5, $d2, $69, $b4), ($ff, $88, $88, $88, $ff, $88, $88, $88), (
  $18, $24, $42, $81, $81, $42, $24, $18), ($cc, $33, $cc, $33, $cc
  , $33, $cc, $33), ($80, $00, $08, $00, $80, $00, $08, $00), ($88,
  $00, $22, $00, $88, $00, $22, $00), (0, 0, 0, 0, 0, 0, 0, 0))
```

Table with standard fill patterns.

```
G1024x768x16 = 30
```

Mode: Resolution 1024x768, 16 colors.

```
G1024x768x16M = 25
```

Mode: Resolution 1024x768, 16M colors.

```
G1024x768x16M32 = 36
```

Mode: Resolution 1024x758, 16M 32-bit colors.

```
G1024x768x256 = 12
```

Mode: Resolution 1024x768, 256 colors.

```
G1024x768x32K = 23
```

Mode: Resolution 1024x768, 32K colors.

```
G1024x768x64K = 24
```

Mode: Resolution 1024x768, 64K colors.

```
G1152x864x16 = 38
```

Mode: Resolution 1152x864, 16 colors.

```
G1152x864x16M = 42
```

Mode: Resolution 1152x864, 16M colors.

```
G1152x864x16M32 = 43
```

Mode: Resolution 1152x864, 16M 32-bitcolors.

```
G1152x864x256 = 39
```

Mode: Resolution 1152x864, 256 colors.

```
G1152x864x32K = 40
```

Mode: Resolution 1152x864, 32K colors.

```
G1152x864x64K = 41
```

Mode: Resolution 1152x864, 64K colors.

```
G1280x1024x16 = 31
```

Mode: Resolution 1280x1024, 16 colors.

```
G1280x1024x16M = 28
```

Mode: Resolution 1280x1024, 16M colors.

```
G1280x1024x16M32 = 37
```

Mode: Resolution 1280x1024, 16M 32-bit colors.

```
G1280x1024x256 = 13
```

Mode: Resolution 1280x1024, 256 colors.

```
G1280x1024x32K = 26
```

Mode: Resolution 1280x1024, 32K colors.

```
G1280x1024x64K = 27
```

Mode: Resolution 1280x1024, 64K colors.

```
G1600x1200x16 = 44
```

Mode: Resolution 1600x1200, 16 colors.

```
G1600x1200x16M = 48
```

Mode: Resolution 1600x1200, 16M colors.

```
G1600x1200x16M32 = 49
```

Mode: Resolution 1600x1200, 16M 32-bit colors.

```
G1600x1200x256 = 45
```

Mode: Resolution 1600x1200, 256 colors.

```
G1600x1200x32K = 46
```

Mode: Resolution 1600x1200, 32K colors.

```
G1600x1200x64K = 47
```

Mode: Resolution 1600x1200, 64K colors.

```
G320x200x16 = 1
```

Mode: Resulution 320x200, 16 colors.

```
G320x200x16M = 16
```

Mode: Resolution 320x200, 16M colors.

```
G320x200x16M32 = 33
```

Mode: Resolution 320x200, 16M 32-bit colors.

```
G320x200x256 = 5
```

Mode: Resolution 320x200, 256 colors.

```
G320x200x32K = 14
```

Mode: Resolution 320x200, 32K colors.

```
G320x200x64K = 15
```

Mode: Resolution 320x200, 64K colors.

```
G320x240x256 = 6
```

Mode: Resolution 320x240, 256 colors.

```
G320x400x256 = 7
```

Mode: Resolution 320x400, 256 colors.

```
G360x480x256 = 8
```

Mode: Resolution 360x480, 256 colors.

```
G640x200x16 = 2
```

Mode: Resolution x, colors.

```
G640x350x16 = 3
```

Mode: Resolution x, colors.

```
G640x480x16 = 4
```

Mode: Resolution x, colors.

```
G640x480x16M = 19
```

Mode: Resolution 640x480, 16M colors.

```
G640x480x16M32 = 34
```

Mode: Resolution 640x480, 16M 32-bit colors.

```
G640x480x2 = 9
```

Mode: Resolution 640x480, 2 colors.

```
G640x480x256 = 10
```

Mode: Resolution 640x480, 256 colors.

```
G640x480x32K = 17
```

Mode: Resolution 640x480, 32K colors.

```
G640x480x64K = 18
```

Mode: Resolution 640x480, 64K colors.

```
G720x348x2 = 32
```

Mode: Resolution 720x348, 2 colors.

```
G800x600x16 = 29
```

Mode: Resolution 800x600, 16 colors.

```
G800x600x16M = 22
```

Mode: Resolution 800x600, 16M colors.

```
G800x600x16M32 = 35
```

Mode: Resolution 800x600, 16M 32-bit colors.

```
G800x600x256 = 11
```

Mode: Resolution 800x600, 256 colors.

```
G800x600x32K = 20
```

Mode: Resolution 800x600, 32K colors.

```
G800x600x64K = 21
```

Mode: Resolution 800x600, 64K colors.

```
GothicFont = 4
```

Font number: Gothic font.

```
GraphStringTransTable : PCharsetTransTable = Nil
```

Table used when transliterating strings.

```
green = 2
```

Color code: green.

```
grError = - 11
```

Error: Unknown error.

```
grFileNotFound = - 3
```

Error: File for driver not found.

```
grFontNotFound = - 8
```

Error: font description file not found.

```
grInvalidDriver = - 4
```

Error: Invalid driver specified.

```
grInvalidFont = - 13
```

Error: Invalid font description.

```
grInvalidFontNum = - 14
```

Error: Invalid font number.

```
grInvalidMode = - 10
```

Error: Invalid mode specified.

```
grInvalidVersion = - 18
```

Error: Invalid version.

```
grIOerror = - 12
```

Error: Unspecified Input/Output error.

```
grNoFloodMem = - 7
```

Error: Could not allocate memory for flood operation.

```
grNoFontMem = -9
```

Error: Not enough memory to load font.

```
grNoInitGraph = -1
```

Error: Graphical system not initialized.

```
grNoLoadMem = -5
```

Error: Memory error.

```
grNoScanMem = -6
```

Error: Could not allocate memory for scan.

```
grNotDetected = -2
```

Error: Graphics device not detected.

```
grOk = 0
```

Graphical operation went OK.

```
HatchFill = 7
```

Fill style: Hatch lines.

```
HercMono = 7
```

Mode: Hercules, mono color.

```
HercMonoHi = 0
```

Mode: Hercules card, monochrome, high resolution.

```
highNewDriver = 21
```

Mode: highest number for new driver.

```
highNewMode = m2048x1536
```

Mode: Highest possible value of the new modes.

```
HorizDir = 0
```

Text write direction: Horizontal.

```
InterleaveFill = 9
```

Fill style: Interleaving lines.

```
LCOMFont = 8
```

Font number: ?

```
LeftText = 0
```

Horizontal text alignment: Align text left.

```
lightblue = 9
```

Color code: Light blue.

```
lightcyan = 11
```

Color code: Light cyan.

```
lightgray = 7
```

Color code: Light gray.

```
lightgreen = 10
```

Color code: Light green.

```
lightmagenta = 13
```

Color code: Light magenta.

```
lightred = 12
```

Color code: Light red.

```
LineFill = 2
```

Fill style: Fill using horizontal lines.

```
lowNewDriver = 11
```

Mode: lowest number for new driver.

```
lowNewMode = m320x200
```

Mode: Lowest possible value of the new modes.

```
LowRes = 6
```

Mode: Low resolution.

```
LtBkSlashFill = 6
```

Fill style: Light diagonal (backslash) lines.

```
LtSlashFill = 3
```

Fill style: Light diagonal (slash) lines.

```
m1024x768 = detectMode + 12
```

Mode: Resolution 1024x768.

```
m1280x1024 = detectMode + 13
```

Mode: Resolution 1280x1024.

```
m1600x1200 = detectMode + 14
```

Mode: Resolution 1600x1200.

```
m2048x1536 = detectMode + 15
```

Mode: Resolution 2048x1536.

```
m320x200 = detectMode + 1
```

Mode: Resolution 320x200.

```
m320x256 = detectMode + 2
```

Mode: Resolution 320x256.

```
m320x400 = detectMode + 3
```

Mode: Resolution 320x400.

```
m512x384 = detectMode + 4
```

Mode: Resolution 512x384.

```
m640x200 = detectMode + 5
```

Mode: Resolution 640x200.

```
m640x256 = detectMode + 6
```

Mode: Resolution 640x256.

```
m640x350 = detectMode + 7
```

Mode: Resolution 640x350.

```
m640x400 = detectMode + 8
```

Mode: Resolution 640x400.

```
m640x480 = detectMode + 9
```

Mode: Resolution 640x480.

```
m800x600 = detectMode + 10
```

Mode: Resolution 800x600.

```
m832x624 = detectMode + 11
```

Mode: Resolution 832x624.

```
magenta = 5
```

Color code: Magenta.

```
MaxColors = 255
```

Max amount of colors in a palette.

```
maxsmallint = high(smallint)
```

Maximum value for smallint type.

```
MCGA = 2
```

Graphic driver for MCGA cards.

```
MCGAC0 = 0
```

MCGA Graphic driver mode C0.

```
MCGAC1 = 1
```

MCGA Graphic driver mode C1.

```
MCGAC2 = 2
```

MCGA Graphic driver mode C2.

```
MCGAC3 = 3
```

MCGA Graphic driver mode C3.

```
MCGAHi = 5
```

MCGA Graphic driver high resolution mode.

```
MCGAMed = 4
```

MCGA Graphic driver medium resolution mode.

```
NormalPut = 0
```

Draw operation: Use Normal (copy) operation.

```
NormWidth = 1
```

Line width: Normal width.

```
NotPut = 4
```

Draw operation: use NOT.

```
OrPut = 2
```

Draw operation: use OR.

```
red = 4
```

Color code: Red.

```
resolutions : Array[lowNewMode..highNewMode] of TResolutionRec =
  ((x: 320; y: 200), (x: 320; y: 256), (x: 320; y: 400), (x: 512; y
  : 384), (x: 640; y: 200), (x: 640; y: 256), (x: 640; y: 350), (x:
  640; y: 400), (x: 640; y: 480), (x: 800; y: 600), (x: 832; y: 624
  ), (x: 1024; y: 768), (x: 1280; y: 1024), (x: 1600; y: 1200), (x:
  2048; y: 1536))
```

Array with actual resolutions of the new modes.

```
RightText = 2
```

Horizontal text alignment: Align text right.

```
SansSerifFont = 3
```

Font number: Sans Serif font.

```
ScriptFont = 5
```

Font number: Script font.

```
SimpleFont = 6
```

Font number: Simple font.

```
SlashFill = 4
```

Fill style: Diagonal (slash) lines.

```
SmallFont = 2
```

Font number: Small font.

```
SolidFill = 1
```

Fill style: Solid fill.

```
SolidLn = 0
```

Line style: Solid line.

```
ThickWidth = 3
```

Line width: double width.

```
TopOff = False
```

Top off.

```
TopOn = True
```

Top on.

```
TopText = 2
```

Vertical text alignment: Align text to top.

```
TriplexFont = 1
```

Font number: Triplex font.

```
TSCRFont = 7
```

Font number: Terminal font.

```
UserBitLn = 4
```

Line style: User defined.

```
UserCharSize = 0
```

User character size.

```
UserFill = 12
```

Fill style: User-defined fill.

```
VertDir = 1
```

Text write direction: Vertical.

```
VESA = 10
```

Mode: VESA graphics adaptor.

```
VGA = 9
```

Mode: VGA graphics adaptor.

```
VGAHi = 2
```

Mode: VGA high resolution (640x480).

```
VGALo = 0
```

Mode: VGA low resolution (640x200).

```
VGAMed = 1
```

Mode: VGA medium resolution (640x350).

```
white = 15
```

Color code: White.

```
WideDotFill = 10
```

Fill style: Widely spaced dotted lines.

```
XHatchFill = 8
```

Fill style: Heavy hatch lines.

```
XORPut = 1
```

Draw operation: use XOR.

```
yellow = 14
```

Color code: Yellow.

56.13.2 Types

```
CircleProc = procedure(X: SmallInt; Y: SmallInt; Radius: Word)
```

Standard circle drawing routine prototype.

```
clrviewproc = procedure
```

Standard clearviewport routine prototype.

```
ColorType = Word
```

Color type alias.

```
defpixelproc = procedure(X: SmallInt; Y: SmallInt)
```

This is the standard putpixel routine used by all function drawing routines, it will use the viewport settings, as well as clip, and use the current foreground color to plot the desired pixel.

```
ellipseproc = procedure(X: SmallInt; Y: SmallInt; XRadius: Word;
  YRadius: Word; stAngle: Word; EndAngle: Word
  ;
                        fp: patternlineproc)
```

Standard ellipse drawing routine prototype.

```
FillPatternType = Array[1..8] of Byte
```

Bit pattern used when drawing lines. Set bits are drawn.

```
GetBkColorProc = function : ColorType
```

GetBkColorProc is the procedure prototype for the GetBkColor (893) method handler in TModeInfo (916). The function should return the color code of the background color.

```
getimageproc = procedure(X1: SmallInt; Y1: SmallInt; X2: SmallInt
  ;
                         Y2: SmallInt; var Bitmap)
```

Standard GetImage (893) procedure prototype.

```
getpixelproc = function(X: SmallInt; Y: SmallInt) : ColorType
```

Standard pixel fetching routine prototype.

```
getrgbpaletteproc = procedure(ColorNum: SmallInt;
  var RedValue: SmallInt;
  var GreenValue: SmallInt;
                                var
  BlueValue: SmallInt)
```

This routine prototype is a hook for GetRGBPalette (893).

```
getscanlineproc = procedure(X1: SmallInt; X2: SmallInt; Y: SmallInt
  ;
                            var data)
```

This routine is used for FloodFill (897) It returns an entire screen scan line with a word for each pixel in the scanline. Also handy for GetImage.

```
graphfreememprc = procedure(var P: Pointer; size: Word)
```

Procedure prototype, used when heap memory is freed by the graph routines.

```
graphgetmemprc = procedure(var P: pointer; size: Word)
```

Procedure prototype, used when heap memory is needed by the graph routines.

```
graph_float = single
```

The platform's preferred floating point size for fast graph operations.

```
hlineproc = procedure(x: SmallInt; x2: SmallInt; y: SmallInt)
```

Standard procedure prototype to draw a single horizontal line.

```
imagesizeproc = function(X1: SmallInt; Y1: SmallInt; X2: SmallInt
  ;
                         Y2: SmallInt) : LongInt
```

Standard ImageSize (893) calculation procedure prototype.

```
initmodeproc = procedure
```

Standard routine prototype to initialize a mode.

```
lineproc = procedure(X1: SmallInt; Y1: SmallInt; X2: SmallInt;
  Y2: SmallInt)
```

Standard line drawing routine prototype.

```
OutTextXYProc = procedure(x: SmallInt; y: SmallInt;
  const TextString: string)
```

This routine prototype is a hook for OutTextXY (894).

```
patternlineproc = procedure(x1: SmallInt; x2: SmallInt; y: SmallInt
  )
```

Standard procedure prototype to draw a patterned line.

```
PCharsetTransTable = ^TCharsetTransTable
```

Pointer to TCharsetTransTable (892) array.

```
PModeInfo = ^TModeInfo
```

Pointer to TModeInfo (916) record.

```
putimageproc = procedure(X: SmallInt; Y: SmallInt; var Bitmap;
  BitBlt: Word)
```

Standard PutImage (894) procedure prototype.

```
putpixelproc = procedure(X: SmallInt; Y: SmallInt; Color: ColorType
  )
```

Standard pixel drawing routine prototype.

```
restorestateproc = procedure
```

Standard routine prototype to restore the graphical state at a closegraph call.

```
savestateproc = procedure
```

Standard routine prototype to save the graphical state before a mode is set.

```
setactivepageproc = procedure(page: Word)
```

Standard SetActivePage (894) procedure prototype.

```
SetAllPaletteProc = procedure(const Palette: PaletteType)
```

This routine prototype is a hook for SetAllPalette (894).

```
SetBkColorProc = procedure(ColorNum: ColorType)
```

`SetBkColorProc` is the procedure prototype for the SetBkColor (894) method handler in TModeInfo (916). The procedure gets passed the color code for the color to set as background color.

```
setrgbpaletteproc = procedure(ColorNum: SmallInt; RedValue: SmallInt
  ;
                              GreenValue: SmallInt; BlueValue:
  SmallInt)
```

This routine prototype is a hook for SetRGBPalette (895).

```
setvisualpageproc = procedure(page: Word)
```

Standard SetVisualPage (895) procedure prototype.

```
TCharsetTransTable = Array[Char] of Char
```

Character transliteration table, with entries for 256 characters.

```
vlineproc = procedure(x: SmallInt; y: SmallInt; y2: SmallInt)
```

Standard procedure prototype to draw a single vertical line.

56.13.3 Variables

```
Circle : CircleProc
```

`Circle` draws a complete circle with center at `(X,Y)`, radius `radius`.

```
ClearViewPort : clrviewproc
```

Clears the current viewport. The current background color is used as filling color. The pointer is set at `(0,0)`.

`DirectPutPixel : defpixelproc`

Hook to directly draw a pixel on the screen.

`GetBkColor : GetBkColorProc`

`GetBkColor` returns the current background color (the palette entry).

`GetImage : getimageproc`

`GetImage` Places a copy of the screen area `(X1,Y1)` to `X2,Y2` in `BitMap`

`GetPixel : getpixelproc`

`GetPixel` returns the color of the point at `(X,Y)`

`GetRGBPalette : getrgbpaletteproc`

Hook to set a RGB palette entries.

`GetScanLine : getscanlineproc`

Hook to get a scan line from the screen.

`GraphFreeMemPtr : graphfreememprc`

Hook to free heap memory.

`GraphGetMemPtr : graphgetmemprc`

Hook to get heap memory.

`HLine : hlineproc`

Hook to draw a solid horizontal line.

`ImageSize : imagesizeproc`

`ImageSize` returns the number of bytes needed to store the image in the rectangle defined by `(X1,Y1)` and `(X2,Y2)`.

`InternalEllipse : ellipseproc`

Hook to draw an ellipse.

`Line : lineproc`

Line draws a line starting from (X1,Y1 to (X2,Y2), in the current line style and color. The current position is put to (X2,Y2)

OutTextXY : OutTextXYProc

OutText puts TextString on the screen, at position (X,Y), using the current font and text settings. The current position is moved to the end of the text.

PatternLine : patternlineproc

Hook to draw a patterned line.

PutImage : putimageproc

PutImage Places the bitmap in Bitmap on the screen at (X1,Y1). How determines how the bitmap will be placed on the screen. Possible values are:

- CopyPut
- XORPut
- ORPut
- AndPut
- NotPut

PutPixel : putpixelproc

Puts a point at (X,Y) using color Color

RestoreVideoState : restorestateproc

Hook to restore a saved video mode.

SaveVideoState : savestateproc

Hook to save the current video state.

SetActivePage : setactivepageproc

Sets Page as the active page for all graphical output.

SetAllPalette : SetAllPaletteProc

Sets the current palette to Palette. Palette is an untyped variable, usually pointing to a record of type PaletteType

SetBkColor : SetBkColorProc

Sets the background color to Color.

SetRGBPalette : setrgbpaletteproc

SetRGBPalette sets the ColorNr-th entry in the palette to the color with RGB-values Red, Green Blue.

SetVisualPage : setvisualpageproc

SetVisualPage sets the video page to page number Page.

VLine : vlineproc

Hook to draw a solid vertical line.

56.14 Procedures and functions

56.14.1 Arc

Synopsis: Draw part of a circle.

Declaration:
```
procedure Arc(X: SmallInt; Y: SmallInt; StAngle: Word; EndAngle: Word;
              Radius: Word)
```

Visibility: default

Description: Arc draws part of a circle with center at (X,Y), radius radius, starting from angle start, stopping at angle stop. These angles are measured counterclockwise.

Errors: None.

See also: Circle (892), Ellipse (897), GetArcCoords (898), PieSlice (907), Sector (908)

56.14.2 Bar

Synopsis: Draw filled rectangle.

Declaration:
```
procedure Bar(x1: SmallInt; y1: SmallInt; x2: SmallInt; y2: SmallInt)
```

Visibility: default

Description: Draws a rectangle with corners at (X1,Y1) and (X2,Y2) and fills it with the current color and fill-style.

Errors: None.

See also: Bar3D (895), Rectangle (907)

56.14.3 Bar3D

Synopsis: Draw filled 3-dimensional rectangle.

Declaration:
```
procedure Bar3D(x1: SmallInt; y1: SmallInt; x2: SmallInt; y2: SmallInt;
                depth: Word; top: Boolean)
```

Visibility: default

Description: Bar3d draws a 3-dimensional Bar with corners at (X1,Y1) and (X2,Y2) and fills it with the current color and fill-style. Depth specifies the number of pixels used to show the depth of the bar.

If Top is true; then a 3-dimensional top is drawn.

Errors: None.

See also: Bar (895), Rectangle (907)

56.14.4 ClearDevice

Synopsis: Clear the complete screen.

Declaration: `procedure ClearDevice`

Visibility: default

Description: Clears the graphical screen (with the current background color), and sets the pointer at `(0,0)`.

Errors: None.

See also: ClearViewPort (893), SetBkColor (894)

56.14.5 Closegraph

Synopsis: Close graphical system.

Declaration: `procedure Closegraph`

Visibility: default

Description: Closes the graphical system, and restores the screen modus which was active before the graphical modus was activated.

Errors: None.

See also: InitGraph (904)

56.14.6 DetectGraph

Synopsis: Detect correct graphical driver to use.

Declaration:
```
procedure DetectGraph(var GraphDriver: SmallInt;
                      var GraphMode: SmallInt)
```

Visibility: default

Description: `DetectGraph` checks the hardware in the PC and determines the driver and screen-modus to be used. These are returned in `Driver` and `Modus`, and can be fed to `InitGraph`. See the `InitGraph` for a list of drivers and modi.

Errors: None.

See also: InitGraph (904)

56.14.7 DrawPoly

Synopsis: Draw a polygon.

Declaration: `procedure DrawPoly(NumPoints: Word; var polypoints)`

Visibility: default

Description: `DrawPoly` draws a polygon with `NumberOfPoints` corner points, using the current color and line-style. PolyPoints is an array of type PointType (915).

Errors: None.

See also: Bar (895), Bar3D (895), Rectangle (907)

56.14.8 Ellipse

Synopsis: Draw an ellipse.

Declaration:
```
procedure Ellipse(X: SmallInt; Y: SmallInt; stAngle: Word;
                  EndAngle: Word; XRadius: Word; YRadius: Word)
```

Visibility: default

Description: `Ellipse` draws part of an ellipse with center at `(X,Y)`. `XRadius` and `Yradius` are the horizontal and vertical radii of the ellipse. `Start` and `Stop` are the starting and stopping angles of the part of the ellipse. They are measured counterclockwise from the X-axis (3 o'clock is equal to 0 degrees). Only positive angles can be specified.

Errors: None.

See also: Arc (895), Circle (892), FillEllipse (897)

56.14.9 FillEllipse

Synopsis: Draw and fill an ellipse.

Declaration:
```
procedure FillEllipse(X: SmallInt; Y: SmallInt; XRadius: Word;
                      YRadius: Word)
```

Visibility: default

Description: `Ellipse` draws an ellipse with center at `(X,Y)`. `XRadius` and `Yradius` are the horizontal and vertical radii of the ellipse. The ellipse is filled with the current color and fill-style.

Errors: None.

See also: Arc (895), Circle (892), GetArcCoords (898), PieSlice (907), Sector (908)

56.14.10 FillPoly

Synopsis: Draw, close and fill a polygon.

Declaration:
```
procedure FillPoly(NumPoints: Word; var PolyPoints)
```

Visibility: default

Description: `FillPoly` draws a polygon with `NumberOfPoints` corner points and fills it using the current color and line-style. PolyPoints is an array of type `PointType`.

Errors: None.

See also: Bar (895), Bar3D (895), Rectangle (907)

56.14.11 FloodFill

Synopsis: Fill an area with a given color.

Declaration:
```
procedure FloodFill(x: SmallInt; y: SmallInt; Border: ColorType)
```

Visibility: default

Description: Fills the area containing the point `(X,Y)`, bounded by the color `BorderColor`.

Errors: None

See also: SetColor (909), SetBkColor (894)

56.14.12 GetArcCoords

Synopsis: Return coordinates of last drawn arc or ellipse.

Declaration: `procedure GetArcCoords(var ArcCoords: ArcCoordsType)`

Visibility: default

Description: `GetArcCoords` returns the coordinates of the latest `Arc` or `Ellipse` call.

Errors: None.

See also: Arc (895), Ellipse (897)

56.14.13 GetAspectRatio

Synopsis: Return screen resolution.

Declaration: `procedure GetAspectRatio(var Xasp: Word; var Yasp: Word)`

Visibility: default

Description: `GetAspectRatio` determines the effective resolution of the screen. The aspect ration can then be calculated as `Xasp/Yasp`.

Errors: None.

See also: InitGraph (904), SetAspectRatio (908)

56.14.14 GetColor

Synopsis: Return current drawing color.

Declaration: `function GetColor : ColorType`

Visibility: default

Description: `GetColor` returns the current drawing color (the palette entry).

Errors: None.

See also: GetColor (898), SetBkColor (894)

56.14.15 GetDefaultPalette

Synopsis: Return default palette.

Declaration: `procedure GetDefaultPalette(var Palette: PaletteType)`

Visibility: default

Description: `GetDefaultPalette` returns the current palette in `Palette`.

Errors: None.

See also: GetColor (898), GetBkColor (893)

56.14.16 GetDirectVideo

Synopsis: Determine whether direct video mode is active.

Declaration: `function GetDirectVideo : Boolean`

Visibility: default

Description: Determine whether direct video mode is active.

56.14.17 GetDriverName

Synopsis: Return current driver name.

Declaration: `function GetDriverName : string`

Visibility: default

Description: `GetDriverName` returns a string containing the name of the current driver.

Errors: None.

See also: GetModeName (901), InitGraph (904)

56.14.18 GetFillPattern

Synopsis: Return current fill pattern.

Declaration: `procedure GetFillPattern(var FillPattern: FillPatternType)`

Visibility: default

Description: `GetFillPattern` returns an array with the current fill-pattern in `FillPattern`

Errors: None

See also: SetFillPattern (909)

56.14.19 GetFillSettings

Synopsis: Return current fill settings.

Declaration: `procedure GetFillSettings(var Fillinfo: FillSettingsType)`

Visibility: default

Description: `GetFillSettings` returns the current fill-settings in `FillInfo`

Errors: None.

See also: SetFillPattern (909)

56.14.20 GetGraphMode

Synopsis: Get current graphical modus.

Declaration: `function GetGraphMode : SmallInt`

Visibility: default

Description: `GetGraphMode` returns the current graphical modus

Errors: None.

See also: InitGraph (904)

56.14.21 GetLineSettings

Synopsis: Get current line drawing settings.

Declaration: `procedure GetLineSettings(var ActiveLineInfo: LineSettingsType)`

Visibility: default

Description: `GetLineSettings` returns the current Line settings in `LineInfo`

Errors: None.

See also: SetLineStyle (910)

56.14.22 GetMaxColor

Synopsis: return maximum number of colors.

Declaration: `function GetMaxColor : ColorType`

Visibility: default

Description: `GetMaxColor` returns the maximum color-number which can be set with `SetColor`. Contrary to Turbo Pascal, this color isn't always guaranteed to be white (for instance in 256+ color modes).

Errors: None.

See also: SetColor (909), GetPaletteSize (902)

56.14.23 GetMaxMode

Synopsis: Return biggest mode for the current driver.

Declaration: `function GetMaxMode : SmallInt`

Visibility: default

Description: `GetMaxMode` returns the highest modus for the current driver.

Errors: None.

See also: InitGraph (904)

56.14.24 GetMaxX

Synopsis: Return maximal X coordinate.

Declaration:
```
function GetMaxX : SmallInt
```

Visibility: default

Description: `GetMaxX` returns the maximum horizontal screen length

Errors: None.

See also: GetMaxY (901)

56.14.25 GetMaxY

Synopsis: Return maximal Y coordinate.

Declaration:
```
function GetMaxY : SmallInt
```

Visibility: default

Description: `GetMaxY` returns the maximum number of screen lines

Errors: None.

See also: GetMaxY (901)

56.14.26 GetModeName

Synopsis: Return description a modus.

Declaration:
```
function GetModeName(ModeNumber: SmallInt) : string
```

Visibility: default

Description: `GetModeName` Returns a string with the name of modus `Modus`

Errors: None.

See also: GetDriverName (899), InitGraph (904)

56.14.27 GetModeRange

Synopsis: Return lowest and highest modus of current driver.

Declaration:
```
procedure GetModeRange(GraphDriver: SmallInt; var LoMode: SmallInt;
                       var HiMode: SmallInt)
```

Visibility: default

Description: `GetModeRange` returns the Lowest and Highest modus of the currently installed driver. If no modes are supported for this driver, HiModus will be -1.

Errors: None.

See also: InitGraph (904)

56.14.28 GetPalette

Synopsis: Return current palette.

Declaration: `procedure GetPalette(var Palette: PaletteType)`

Visibility: default

Description: `GetPalette` returns in `Palette` the current palette.

Errors: None.

See also: GetPaletteSize (902), SetPalette (911)

56.14.29 GetPaletteSize

Synopsis: Return maximal number of entries in current palette.

Declaration: `function GetPaletteSize : SmallInt`

Visibility: default

Description: `GetPaletteSize` returns the maximum number of entries in the current palette.

Errors: None.

See also: GetPalette (902), SetPalette (911)

56.14.30 GetTextSettings

Synopsis: Return current text style.

Declaration: `procedure GetTextSettings(var TextInfo: TextSettingsType)`

Visibility: default

Description: `GetTextSettings` returns the current text style settings : The font, direction, size and placement as set with `SetTextStyle` and `SetTextJustify`

Errors: None.

See also: SetTextStyle (911), SetTextJustify (911)

56.14.31 GetViewSettings

Synopsis: Return current viewport.

Declaration: `procedure GetViewSettings(var viewport: ViewPortType)`

Visibility: default

Description: `GetViewSettings` returns the current viewport and clipping settings in `ViewPort`.

Errors: None.

See also: SetViewPort (912)

56.14.32 GetX

Synopsis: Return current cursor X position.

Declaration: `function GetX : SmallInt`

Visibility: default

Description: `GetX` returns the X-coordinate of the current position of the graphical pointer

Errors: None.

See also: GetY (903)

56.14.33 GetY

Synopsis: Return current cursor Y position.

Declaration: `function GetY : SmallInt`

Visibility: default

Description: `GetY` returns the Y-coordinate of the current position of the graphical pointer

Errors: None.

See also: GetX (903)

56.14.34 GraphDefaults

Synopsis: Reset graphical mode to defaults.

Declaration: `procedure GraphDefaults`

Visibility: default

Description: `GraphDefaults` resets all settings for viewport, palette, foreground and background pattern, line-style and pattern, filling style, filling color and pattern, font, text-placement and text size.

Errors: None.

See also: SetViewPort (912), SetFillStyle (909), SetColor (909), SetBkColor (894), SetLineStyle (910)

56.14.35 GraphErrorMsg

Synopsis: Return a description of an error.

Declaration: `function GraphErrorMsg(ErrorCode: SmallInt) : string`

Visibility: default

Description: `GraphErrorMsg` returns a string describing the error `Errorcode`. This string can be used to let the user know what went wrong.

Errors: None.

See also: GraphResult (904)

56.14.36 GraphResult

Synopsis: Result of last graphical operation.

Declaration: `function GraphResult : SmallInt`

Visibility: default

Description: `GraphResult` returns an error-code for the last graphical operation. If the returned value is zero, all went well. A value different from zero means an error has occurred. besides all operations which draw something on the screen, the following procedures also can produce a `GraphResult` different from zero:

- InstallUserFont (905)
- SetLineStyle (910)
- SetWriteMode (913)
- SetFillStyle (909)
- SetTextJustify (911)
- SetGraphMode (910)
- SetTextStyle (911)

Errors: None.

See also: GraphErrorMsg (903)

56.14.37 InitGraph

Synopsis: Initialize graphical system.

Declaration:
```
procedure InitGraph(var GraphDriver: SmallInt; var GraphMode: SmallInt;
                    const PathToDriver: string)
```

Visibility: default

Description: `InitGraph` initializes the `graph` package. `GraphDriver` has two valid values: `GraphDriver=0` which performs an auto detect and initializes the highest possible mode with the most colors. 1024x768x64K is the highest possible resolution supported by the driver, if you need a higher resolution, you must edit MODES.PPI. If you need another mode, then set `GraphDriver` to a value different from zero and `graphmode` to the mode you wish (VESA modes where 640x480x256 is `101h` etc.). `PathToDriver` is only needed, if you use the BGI fonts from Borland. Free Pascal does not offer BGI fonts like Borland, these must be obtained separately.

Example code:

```
var
   gd,gm : integer;
   PathToDriver : string;
begin
   gd:=detect; { highest possible resolution }
   gm:=0; { not needed, auto detection }
   PathToDriver:='C:\PP\BGI'; { path to BGI fonts,
                                drivers aren't needed }
   InitGraph(gd,gm,PathToDriver);
   if GraphResult<>grok then
     halt; ..... { whatever you need }
   CloseGraph; { restores the old graphics mode }
end.
```

Errors: None.

See also: Modes (867), DetectGraph (896), CloseGraph (896), GraphResult (904)

56.14.38 InstallUserDriver

Synopsis: Install a user driver.

Declaration:
```
function InstallUserDriver(Name: string; AutoDetectPtr: Pointer)
                                  : SmallInt
```

Visibility: default

Description: `InstallUserDriver` adds the device-driver `DriverPath` to the list of .BGI drivers. `AutoDetectPtr` is a pointer to a possible auto-detect function.

Errors: None.

See also: InitGraph (904), InstallUserFont (905)

56.14.39 InstallUserFont

Synopsis: Install a user-defined font.

Declaration:
```
function InstallUserFont(const FontFileName: string) : SmallInt
```

Visibility: default

Description: `InstallUserFont` adds the font in `FontPath` to the list of fonts of the .BGI system.

Errors: None.

See also: InitGraph (904), InstallUserDriver (905)

56.14.40 LineRel

Synopsis: Draw a line starting from current position in given direction.

Declaration:
```
procedure LineRel(Dx: SmallInt; Dy: SmallInt)
```

Visibility: default

Description: `LineRel` draws a line starting from the current pointer position to the point `(DX,DY)`, **relative** to the current position, in the current line style and color. The Current Position is set to the endpoint of the line.

Errors: None.

See also: Line (894), LineTo (906)

56.14.41 LineTo

Synopsis: Draw a line starting from current position to a given point.

Declaration: `procedure LineTo(X: SmallInt; Y: SmallInt)`

Visibility: default

Description: `LineTo` draws a line starting from the current pointer position to the point `(DX,DY)`, **relative** to the current position, in the current line style and color. The Current position is set to the end of the line.

Errors: None.

See also: LineRel (905), Line (894)

56.14.42 MoveRel

Synopsis: Move cursor relative to current position.

Declaration: `procedure MoveRel(Dx: SmallInt; Dy: SmallInt)`

Visibility: default

Description: `MoveRel` moves the pointer to the point `(DX,DY)`, relative to the current pointer position

Errors: None.

See also: MoveTo (906)

56.14.43 MoveTo

Synopsis: Move cursor to absolute position.

Declaration: `procedure MoveTo(X: SmallInt; Y: SmallInt)`

Visibility: default

Description: `MoveTo` moves the pointer to the point `(X,Y)`.

Errors: None.

See also: MoveRel (906)

56.14.44 OutText

Synopsis: Write text on the screen at the current location.

Declaration: `procedure OutText(const TextString: string)`

Visibility: default

Description: `OutText` puts `TextString` on the screen, at the current pointer position, using the current font and text settings. The current position is moved to the end of the text.

Errors: None.

See also: OutTextXY (894)

56.14.45 PieSlice

Synopsis: Draw a pie-slice.

Declaration:
```
procedure PieSlice(X: SmallInt; Y: SmallInt; stangle: SmallInt;
                   endAngle: SmallInt; Radius: Word)
```

Visibility: default

Description: `PieSlice` draws and fills a sector of a circle with center `(X,Y)` and radius `Radius`, starting at angle `Start` and ending at angle `Stop`.

Errors: None.

See also: Arc (895), Circle (892), Sector (908)

56.14.46 queryadapterinfo

Synopsis: Function called to retrieve the current video adapter settings.

Declaration:
```
function queryadapterinfo : PModeInfo
```

Visibility: default

56.14.47 Rectangle

Synopsis: Draw a rectangle on the screen.

Declaration:
```
procedure Rectangle(x1: SmallInt; y1: SmallInt; x2: SmallInt;
                    y2: SmallInt)
```

Visibility: default

Description: Draws a rectangle with corners at `(X1,Y1)` and `(X2,Y2)`, using the current color and style.

Errors: None.

See also: Bar (895), Bar3D (895)

56.14.48 RegisterBGIDriver

Synopsis: Register a new BGI driver.

Declaration:
```
function RegisterBGIDriver(driver: pointer) : SmallInt
```

Visibility: default

Description: Registers a user-defined BGI driver

Errors: None.

See also: InstallUserDriver (905), RegisterBGIFont (908)

56.14.49 RegisterBGIfont

Synopsis: Register a new BGI font.

Declaration:
```
function RegisterBGIfont(font: pointer) : SmallInt
```

Visibility: default

Description: Registers a user-defined BGI driver

Errors: None.

See also: InstallUserFont (905), RegisterBGIDriver (907)

56.14.50 RestoreCrtMode

Synopsis: Restore text screen.

Declaration:
```
procedure RestoreCrtMode
```

Visibility: default

Description: Restores the screen modus which was active before the graphical modus was started.
To get back to the graph mode you were last in, you can use `SetGraphMode(GetGraphMode)`

Errors: None.

See also: InitGraph (904)

56.14.51 Sector

Synopsis: Draw and fill a sector of an ellipse.

Declaration:
```
procedure Sector(x: SmallInt; y: SmallInt; StAngle: Word;
                 EndAngle: Word; XRadius: Word; YRadius: Word)
```

Visibility: default

Description: `Sector` draws and fills a sector of an ellipse with center `(X,Y)` and radii `XRadius` and `YRadius`, starting at angle `Start` and ending at angle `Stop`.

Errors: None.

See also: Arc (895), Circle (892), PieSlice (907)

56.14.52 SetAspectRatio

Synopsis: Set aspect ration of the screen.

Declaration:
```
procedure SetAspectRatio(Xasp: Word; Yasp: Word)
```

Visibility: default

Description: Sets the aspect ratio of the current screen to `Xasp/Yasp`.

Errors: None

See also: InitGraph (904), GetAspectRatio (898)

56.14.53 SetColor

Synopsis: Set foreground drawing color.

Declaration: `procedure SetColor(Color: ColorType)`

Visibility: default

Description: Sets the foreground color to `Color`.

Errors: None.

See also: GetColor (898), SetBkColor (894), SetWriteMode (913)

56.14.54 SetDirectVideo

Synopsis: Attempt to enter direct video mode.

Declaration: `procedure SetDirectVideo(DirectAccess: Boolean)`

Visibility: default

Description: `SetDirectVideo` attempts to enter direct video mode. In that mode, everything is drawn straight in the video buffer.

56.14.55 SetFillPattern

Synopsis: Set drawing fill pattern.

Declaration: `procedure SetFillPattern(Pattern: FillPatternType; Color: ColorType)`

Visibility: default

Description: `SetFillPattern` sets the current fill-pattern to `FillPattern`, and the filling color to `Color` The pattern is an 8x8 raster, corresponding to the 64 bits in `FillPattern`.

Errors: None

See also: GetFillPattern (899), SetFillStyle (909), SetWriteMode (913)

56.14.56 SetFillStyle

Synopsis: Set drawing fill style.

Declaration: `procedure SetFillStyle(Pattern: Word; Color: ColorType)`

Visibility: default

Description: `SetFillStyle` sets the filling pattern and color to one of the predefined filling patterns. `Pattern` can be one of the following predefined constants :

EmptyFillUses backgroundcolor.

SolidFillUses filling color

LineFillFills with horizontal lines.

ltSlashFillFills with lines from left-under to top-right.

SlashFillIdem as previous, thick lines.

BkSlashFillFills with thick lines from left-Top to bottom-right.

LtBkSlashFillIdem as previous, normal lines.

HatchFillFills with a hatch-like pattern.

XHatchFillFills with a hatch pattern, rotated 45 degrees.

InterLeaveFill

WideDotFillFills with dots, wide spacing.

CloseDotFillFills with dots, narrow spacing.

UserFillFills with a user-defined pattern.

Errors: None.

See also: SetFillPattern (909), SetWriteMode (913)

56.14.57 SetGraphMode

Synopsis: Set graphical mode.

Declaration: `procedure SetGraphMode(Mode: SmallInt)`

Visibility: default

Description: `SetGraphMode` sets the graphical mode and clears the screen.

Errors: None.

See also: InitGraph (904)

56.14.58 SetLineStyle

Synopsis: Set line drawing style.

Declaration: `procedure SetLineStyle(LineStyle: Word; Pattern: Word; Thickness: Word)`

Visibility: default

Description: `SetLineStyle` sets the drawing style for lines. You can specify a `LineStyle` which is one of the following predefined constants:

SolidIndraws a solid line.

DottedInDraws a dotted line.

CenterIndraws a non-broken centered line.

DashedIndraws a dashed line.

UserBitInDraws a User-defined bit pattern.

If `UserBitIn` is specified then `Pattern` contains the bit pattern. In all another cases, `Pattern` is ignored. The parameter `Width` indicates how thick the line should be. You can specify one of the following predefined constants:

NormWidthNormal line width

ThickWidthDouble line width

Errors: None.

See also: GetLineSettings (900), SetWriteMode (913)

56.14.59 SetPalette

Synopsis: Set palette entry using color constant.

Declaration: `procedure SetPalette(ColorNum: Word; Color: ShortInt)`

Visibility: default

Description: `SetPalette` changes the `ColorNr`-th entry in the palette to `NewColor`

Errors: None.

See also: SetAllPalette (894), SetRGBPalette (895)

56.14.60 SetTextJustify

Synopsis: Set text placement style.

Declaration: `procedure SetTextJustify(horiz: Word; vert: Word)`

Visibility: default

Description: `SetTextJustify` controls the placement of new text, relative to the (graphical) cursor position. `Horizontal` controls horizontal placement, and can be one of the following predefined constants:

LeftTextText is set left of the pointer.

CenterTextText is set centered horizontally on the pointer.

RightTextText is set to the right of the pointer.

`Vertical` controls the vertical placement of the text, relative to the (graphical) cursor position. Its value can be one of the following predefined constants :

BottomTextText is placed under the pointer.

CenterTextText is placed centered vertically on the pointer.

TopTextText is placed above the pointer.

Errors: None.

See also: OutText (906), OutTextXY (894)

56.14.61 SetTextStyle

Synopsis: Set text style.

Declaration: `procedure SetTextStyle(font: Word; direction: Word; charsize: Word)`

Visibility: default

Description: `SetTextStyle` controls the style of text to be put on the screen. predefined constants for `Font` are:

DefaultFontThe default font

TriplexFontA special font

SmallFontA smaller font

SansSerifFontA sans-serif font (like Arial)

GothicFontA gothic font

ScriptFontA script font

SimpleFontA simple font

TSCRFontTerminal screen font

LCOMFont?

EuroFont?

BoldFontA bold typeface font

predefined constants for `Direction` are :

HorizDirWrite horizontal

VertDirWrite vertical

Errors: None.

See also: GetTextSettings (902)

56.14.62 SetUserCharSize

Synopsis: Set user character size for vector font.

Declaration:
```
procedure SetUserCharSize(Multx: Word; Divx: Word; Multy: Word;
                          Divy: Word)
```

Visibility: default

Description: Sets the width and height of vector-fonts. The horizontal size is given by `Xasp1/Xasp2`, and the vertical size by `Yasp1/Yasp2`.

Errors: None.

See also: SetTextStyle (911)

56.14.63 SetViewPort

Synopsis: Set the graphical drawing window.

Declaration:
```
procedure SetViewPort(X1: SmallInt; Y1: SmallInt; X2: SmallInt;
                      Y2: SmallInt; Clip: Boolean)
```

Visibility: default

Description: Sets the current graphical viewport (window) to the rectangle defined by the top-left corner `(X1,Y1)` and the bottom-right corner `(X2,Y2)`. If `Clip` is true, anything drawn outside the viewport (window) will be clipped (i.e. not drawn). Coordinates specified after this call are relative to the top-left corner of the viewport.

Errors: None.

See also: GetViewSettings (902)

56.14.64 SetWriteMode

Synopsis: Specify binary operation to perform when drawing on screen.

Declaration: `procedure SetWriteMode(WriteMode: SmallInt)`

Visibility: default

Description: `SetWriteMode` controls the drawing of lines on the screen. It controls the binary operation used when drawing lines on the screen. `Mode` can be one of the following predefined constants:

CopyPutDraw as specified using current bitmask and color

XORPutDraw XOR-ing current bitmask and color

Errors: None.

See also: SetColor (909), SetBkColor (894), SetLineStyle (910), SetFillStyle (909)

56.14.65 SetWriteModeEx

Synopsis: Set write mode (extended version).

Declaration: `procedure SetWriteModeEx(WriteMode: SmallInt)`

Visibility: default

Description: `SetWriteModeEx` sets the graph write mode similar to SetWriteMode (913) but only if the `WriteMode` is in the range `CopyPut..NotPut`. If the provided value is outside this range, it is not set.

See also: SetWriteMode (913)

56.14.66 TextHeight

Synopsis: Return height (in pixels) of the given string.

Declaration: `function TextHeight(const TextString: string) : Word`

Visibility: default

Description: `TextHeight` returns the height (in pixels) of the string `S` in the current font and text-size.

Errors: None.

See also: TextWidth (913)

56.14.67 TextWidth

Synopsis: Return width (in pixels) of the given string.

Declaration: `function TextWidth(const TextString: string) : Word`

Visibility: default

Description: `TextHeight` returns the width (in pixels) of the string `S` in the current font and text-size.

Errors: None.

See also: TextHeight (913)

56.15 ArcCoordsType

```
ArcCoordsType = record
  x : SmallInt;
  y : SmallInt;
  xstart :
  SmallInt;
  ystart : SmallInt;
  xend : SmallInt;
  yend : SmallInt
  ;
end
```

Describe the last arc which was drawn on screen.

56.16 FillSettingsType

```
FillSettingsType = record
  pattern : Word;
  color : ColorType;
end
```

Record describing fill mode.

56.17 LineSettingsType

```
LineSettingsType = record
  linestyle : Word;
  pattern : Word;
  thickness : Word;
end
```

Record describing current line drawing mode.

56.18 PaletteType

```
PaletteType = record
  Size : LongInt;
  Colors : Array[0..MaxColors
  ] of RGBRec;
end
```

Record describing palette.

56.19 PointType

```
PointType = record
```

```
  x : SmallInt;
  y : SmallInt;
end
```

Record describing a point in a 2 dimensional plane.

56.20 RGBRec

```
RGBRec = packed record
  Red : SmallInt;
  Green : SmallInt;
  Blue
  : SmallInt;
end
```

Record describing palette RGB color.

56.21 TextSettingsType

```
TextSettingsType = record
  font : Word;
  direction : Word;
  charsize
  : Word;
  horiz : Word;
  vert : Word;
end
```

Record describing how texts are drawn.

56.22 TModeInfo

```
TModeInfo = record
  DriverNumber : SmallInt;
  ModeNumber : SmallInt
  ;
  internModeNumber : SmallInt;
  MaxColor : LongInt;
  PaletteSize
  : LongInt;
  XAspect : Word;
  YAspect : Word;
  MaxX : Word;
  MaxY
  : Word;
  DirectColor : Boolean;
  Hardwarepages : Byte;
  ModeName
```

```
  : string;
  DirectPutPixel : defpixelproc;
  GetPixel : getpixelproc
  ;
  PutPixel : putpixelproc;
  SetRGBPalette : setrgbpaletteproc;
  GetRGBPalette : getrgbpaletteproc;
  SetAllPalette : SetAllPaletteProc
  ;
  SetVisualPage : setvisualpageproc;
  SetActivePage : setactivepageproc
  ;
  ClearViewPort : clrviewproc;
  PutImage : putimageproc;
  GetImage
  : getimageproc;
  ImageSize : imagesizeproc;
  GetScanLine : getscanlineproc
  ;
  Line : lineproc;
  InternalEllipse : ellipseproc;
  PatternLine
  : patternlineproc;
  HLine : hlineproc;
  VLine : vlineproc;
  Circle
  : CircleProc;
  InitMode : initmodeproc;
  OutTextXY : OutTextXYProc
  ;
  SetBKColor : SetBkColorProc;
  GetBKColor : GetBkColorProc;
  next : PModeInfo;
end
```

Record describing a graphical mode.

56.23 TResolutionRec

```
TResolutionRec = record
  x : LongInt;
  y : LongInt;
end
```

Record describing resolution.

56.24 ViewPortType

```
ViewPortType = record
  x1 : SmallInt;
```

```
  y1 : SmallInt;
  x2 : SmallInt
  ;
  y2 : SmallInt;
  Clip : Boolean;
end
```

Record describing a viewport.

Chapter 57

Reference for unit 'heaptrc'

57.1 Used units

Table 57.1: Used units by unit 'heaptrc'

Name	Page
System	1362

57.2 Overview

This document describes the HEAPTRC unit for Free Pascal. It was written by Pierre Muller. It is system independent, and works on all supported systems.

The HEAPTRC unit can be used to debug your memory allocation/deallocation. It keeps track of the calls to getmem/freemem, and, implicitly, of New/Dispose statements.

When the program exits, or when you request it explicitly. It displays the total memory used, and then dumps a list of blocks that were allocated but not freed. It also displays where the memory was allocated.

If there are any inconsistencies, such as memory blocks being allocated or freed twice, or a memory block that is released but with wrong size, this will be displayed also.

The information that is stored/displayed can be customized using some constants.

Do not use this unit directly, instead use the `-gh` switch, to let the compiler insert the unit in the uses clause. You can detect whether the heaptrc unit is actually used in the main program code with the following function:

```
function IsHeapTraceActive : boolean;

begin
{$if declared(UseHeapTrace)}
  Result:=UseHeapTrace;
{$else}
  Result:=False;
{$endif}
end;
```

57.3 Controlling HeapTrc with environment variables.

The HeapTrc unit can be controlled with the HEAPTRC environment variable. The contents of this variable controls the initial setting of some constants in the unit. HEAPTRC consists of one or more of the following strings, separated by spaces:

keepreleased If this string occurs, then the KeepReleased (921) variable is set to True

disabled If this string occurs, then the UseHeapTrace (922) variable is set to False and the heap trace is disabled. It does not make sense to combine this value with other values.

nohalt If this string occurs, then the HaltOnError (920) variable is set to False, so the program continues executing even in case of a heap error.

log=filename If this string occurs, then the output of heaptrc is sent to the specified Filename. (see also SetHeapTraceOutput (924))

The following are valid values for the HEAPTRC variable:

```
HEAPTRC=disabled
HEAPTRC="keepreleased log=heap.log"
HEAPTRC="log=myheap.log nohalt"
```

Note that these strings are case sensitive, and the name of the variable too.

57.4 HeapTrc Usage.

You must use the -gh switch, to let the compiler insert the unit by itself, so you don't have to include it in your uses clause. In fact, as of version 3.0.0 you may no longer do so.

The below example shows how to use the heaptrc unit.

This is the memory dump shown when running this program in a standard way:

```
Marked memory at 0040FA50 invalid
Wrong size : 128 allocated 64 freed
  0x00408708
  0x0040CB49
  0x0040C481
Call trace for block 0x0040FA50 size 128
  0x0040CB3D
  0x0040C481
```

If you use the lineinfo unit (or use the -gl switch) as well, then heaptrc will also give you the filenames and line-numbers of the procedures in the backtrace:

```
Marked memory at 00410DA0 invalid
Wrong size : 128 allocated 64 freed
  0x004094B8
  0x0040D8F9  main,  line 25 of heapex.pp
  0x0040D231
Call trace for block 0x00410DA0 size 128
  0x0040D8ED  main,  line 23 of heapex.pp
  0x0040D231
```

If lines without filename/line-number occur, this means there is a unit which has no debug info included.

Listing: ./examples/heapex/heapex.pp

```
Program heapex;

{
  Program used to demonstrate the usage of heaptrc unit
  Compile this program with the -gh command-line option
}

Var P1 : ^Longint;
    P2 : Pointer;
    I : longint;

begin
  {$IF NOT DECLARED(heaptrc)}
  {$ERROR You mmust compile this program with -gh}
  {$ENDIF}
  New(P1);
  // causes previous allocation not to be de-allocated
  New(P1);
  Dispose(P1);
  For I:=1 to 10 do
    begin
    GetMem (P2,128);
    // When I is even, deallocate block. We loose 5 times 128
    // bytes this way.
    If (I mod 2) = 0 Then FreeMem(P2,128);
    end;
  GetMem(P2,128);
  // This will provoke an error and a memory dump
  Freemem (P2,64);
end.
```

57.5 Constants, types and variables

57.5.1 Constants

```
add_tail : Boolean = True
```

If `add\_tail` is `True` (the default) then a check is also performed on the memory location just behind the allocated memory.

```
GlobalSkipIfNoLeaks : Boolean = False
```

`GlobalSkipIfNoLeaks` will, when set to `true`, disable the output of the heaptrc unit if no memory leaks will detected, i.e. there will only be output if there were actual memory leaks.

```
HaltOnError : Boolean = True
```

If `HaltOnError` is set to `True` then an illegal call to `FreeMem` will cause the memory manager to execute a `halt(1)` instruction, causing a memory dump. By Default it is set to `True`.

```
HaltOnNotReleased : Boolean = False
```

HaltOnNotReleased can be set to True to make the DumpHeap (922) procedure halt (exit code 203) the program if any memory was not released when the dump is made. If it is False (the default) then DumpHeap just returns.

```
keepreleased : Boolean = False
```

If keepreleased is set to true, then a list of freed memory blocks is kept. This is useful if you suspect that the same memory block is released twice. However, this option is very memory intensive, so use it sparingly, and only when it's really necessary.

```
maxprintedblocklength : Integer = 128
```

maxprintedblocklength determines the maximum number of bytes written by a memory block dump, as produced when printleakedblock (921) or printfaultyblock (921) are true. If the size of the memory block is larger than this size, then only the first maxprintedblocklength will be included in the dump.

```
printfaultyblock : Boolean = False
```

printleakedblock can be set to True to print a memory dump of faulty memory blocks (in case a memory override occurs) The block is printed as a series of hexadecimal numbers, representing the bytes in the memory block. At most maxprintedblocklength (921) bytes of the memory block will be printed.

```
printleakedblock : Boolean = False
```

printleakedblock can be set to True to print a memory dump of unreleased blocks when the heaptrc unit produces a summary of memory leaks. The block is printed as a series of hexadecimal numbers, representing the bytes in the memory block. At most maxprintedblocklength (921) bytes of the memory block will be printed.

```
quicktrace : Boolean = True
```

Quicktrace determines whether the memory manager checks whether a block that is about to be released is allocated correctly. This is a rather time consuming search, and slows program execution significantly, so by default it is set to True.

```
tail_size : LongInt = sizeof(ptruint)
```

This is the size of the tail block added to every memory block when add_tail (920) is True.

```
tracesize = 16
```

Tracesize specifies how many levels of calls are displayed of the call stack during the memory dump. If you specify keepreleased:=True then half the TraceSize is reserved for the GetMem call stack, and the other half is reserved for the FreeMem call stack. For example, the default value of 8 will cause eight levels of call frames to be dumped for the getmem call if keepreleased is False. If KeepReleased is true, then 4 levels of call frames will be dumped for the GetMem call and 4 frames will be dumped for the FreeMem call. If you want to change this value, you must recode the heaptrc unit.

```
usecrc : Boolean = True
```

If `usecrc` is `True` (the default) then a crc check is performed on locations before and after the allocated memory. This is useful to detect memory overwrites.

```
useheaptrace : Boolean = True
```

This variable can be set to `False` at program startup to disable the heap trace functionality completely. Because this variable is checked in the initialization section of the unit, the only way to do this is through the help of an environment variable (919), which must be set before the program starts. It follows that setting the variable to `False` while the main program is executed will have no effect other than not displaying the heap usage report at program exit.

Note that this disables the heaptrace functionality completely: the tracing heapmanager is simply not installed if the variable is set to `False`.

57.5.2 Types

```
tdisplayextrainfoProc = procedure(var ptext: text; p: pointer)
```

The `TDisplayExtraInfoType` is a procedural type used in the SetHeapExtraInfo (923) call to display a memory location which was previously filled with TFillExtraInfoProc (922)

```
tFillExtraInfoProc = procedure(p: pointer)
```

The `TFillExtraInfoProc` is a procedural type used in the SetHeapExtraInfo (923) call to fill a memory location with extra data for displaying.

57.6 Procedures and functions

57.6.1 CheckPointer

Synopsis: Check if a pointer is in the address range of the application.

Declaration:
```
procedure CheckPointer(p: pointer)
```

Visibility: default

Description: `CheckPointer` checks if the pointer is in the address range of the application, more specifically, if it is in the heap. if not, it prints an error and stops the program with aruntime error 204.

57.6.2 DumpHeap

Synopsis: Dump memory usage report to stderr.

Declaration:
```
procedure DumpHeap
procedure DumpHeap(SkipIfNoLeaks: Boolean)
```

Visibility: default

Description: `DumpHeap` dumps to standard error a summary of memory usage. It is called automatically by the heaptrc unit when your program exits (by installing an exit procedure), but it can be called at any time.

Errors: None.

57.6.3 SetHeapExtraInfo

Synopsis: Store extra information in blocks.

Declaration:
```
procedure SetHeapExtraInfo(size: PtrUInt; fillproc: tFillExtraInfoProc;
                           displayproc: tdisplayextrainfoProc)
```

Visibility: default

Description: You can use `SetHeapExtraInfo` to store extra info in the blocks that the heaptrc unit reserves when tracing getmem calls. `Size` indicates the size (in bytes) that the trace mechanism should reserve for your extra information. For each call to `getmem`, `FillProc` will be called, and passed a pointer to the memory reserved.

When dumping the memory summary, the extra info is shown by calling `displayproc` and passing it the memory location which was filled by `fillproc`. It should write the information in readable form to the text file provided in the call to `displayproc`

Errors: You can only call `SetHeapExtraInfo` if no memory has been allocated yet. If memory was already allocated prior to the call to `SetHeapExtraInfo`, then an error will be displayed on standard error output, and a DumpHeap (922) is executed.

See also: DumpHeap (922), SetHeapTraceOutput (924)

Listing: ./examples/heapex/setinfo.pp

```
Program heapex;

{ Program used to demonstrate the usage of heaptrc unit }

{ you must compile this program with the -gh command-line switch of FPC}

Var P1 : ^Longint;
    P2 : Pointer;
    I : longint;
    Marker : Longint;

Procedure SetMarker (P : pointer);

Type PLongint = ^Longint;

begin
  PLongint(P)^:=Marker;
end;

Procedure  Part1;

begin
  // Blocks allocated here are marked with $FFAAFFAA = -5570646
  Marker := $FFAAFFAA;
  New(P1);
  New(P1);
  Dispose(P1);
  For I:=1 to 10 do
    begin
    GetMem (P2,128);
    If (I mod 2) = 0 Then FreeMem(P2,128);
    end;
  GetMem(P2,128);
end;
```

```
Procedure  Part2;

begin
  // Blocks allocated here are marked with $FAFAFAFA = -84215046
  Marker := $FAFAFAFA;
  New(P1);
  New(P1);
  Dispose(P1);
  For I:=1 to 10 do
    begin
    GetMem (P2,128);
    If (I mod 2) = 0 Then FreeMem(P2,128);
    end;
  GetMem(P2,128);
end;

begin
 SetExtraInfo(SizeOf(Marker),@SetMarker);
 Writeln ('Part 1');
 part1;
 Writeln('Part 2');
 part2;
end.
```

57.6.4 SetHeapTraceOutput

Synopsis: Specify filename or text file for heap trace output.

Declaration:
```
procedure SetHeapTraceOutput(const name: string);  Overload
procedure SetHeapTraceOutput(var ATextOutput: Text);  Overload
```

Visibility: default

Description: `SetHeapTraceOutput` using the `name` argument sets the filename into which heap trace info will be written. You can also use the `ATextOutput` form, where you pass an initialized text file record, this record will then be used to write the heap trace.

By default information is written to standard error, this function allows you to redirect the information to a file with full filename `name` or a text file `ATextOutput`.

Errors: If the file cannot be written to, errors will occur when writing the trace.

See also: SetHeapExtraInfo (923)

Chapter 58

Reference for unit 'ipc'

58.1 Used units

Table 58.1: Used units by unit 'ipc'

Name	Page
BaseUnix	146
System	1362
unixtype	2175

58.2 Overview

This document describes the IPC unit for Free Pascal. It was written for Linux by Michael Van Canneyt. It gives all the functionality of System V Inter-Process Communication: shared memory, semaphores and messages. It works only on the Linux operating system.

Many constants here are provided for completeness only, and should under normal circumstances not be used by the programmer.

58.3 Constants, types and variables

58.3.1 Constants

```
IPC_CREAT = 1 shl 9
```

Create if key is nonexistent.

```
IPC_EXCL = 2 shl 9
```

fail if key exists.

```
IPC_INFO = 3
```

For ipcs call.

```
IPC_NOWAIT = 4 shl 9
```

return error on wait.

```
IPC_PRIVATE = TKey(0)
```

```
IPC_RMID = 0
```

Remove resource.

```
IPC_SET = 1
```

set ipc_perm options.

```
IPC_STAT = 2
```

get ipc_perm options.

```
MSGMAX = 4056
```

Internal Message control code. Do not use.

```
MSGMNB = 16384
```

Internal Message control code. Do not use.

```
MSGMNI = 128
```

Internal Message control code. Do not use.

```
MSG_EXCEPT = 2 shl 12
```

Internal Message control code. Do not use.

```
MSG_NOERROR = 1 shl 12
```

Internal Message control code. Do not use.

```
SEM_GETALL = 13
```

Semaphore operation: Get all semaphore values.

```
SEM_GETNCNT = 14
```

Semaphore operation: Get number of processes waiting for resource.

```
SEM_GETPID = 11
```

Semaphore operation: Get process ID of last operation.

```
SEM_GETVAL = 12
```

Semaphore operation: Get current value of semaphore.

```
SEM_GETZCNT = 15
```

Semaphore operation: Get number of processes waiting for semaphores to reach zero.

```
SEM_SEMMNI = 128
```

Semaphore operation: ?

```
SEM_SEMMNS = SEM_SEMMNI * SEM_SEMMSL
```

Semaphore operation: ?

```
SEM_SEMMSL = 32
```

Semaphore operation: ?

```
SEM_SEMOPM = 32
```

Semaphore operation: ?

```
SEM_SEMVMX = 32767
```

Semaphore operation: ?

```
SEM_SETALL = 17
```

Semaphore operation: Set all semaphore values.

```
SEM_SETVAL = 16
```

Semaphore operation: Set semaphore value.

```
SEM_UNDO = $1000
```

Constant for use in semop (939).

```
SHM_LOCK = 11
```

This constant is used in the shmctl (941) call.

```
SHM_R = 4 shl 6
```

This constant is used in the shmctl (941) call.

```
SHM_RDONLY = 1 shl 12
```

This constant is used in the shmctl (941) call.

```
SHM_REMAP = 4 shl 12
```

This constant is used in the shmctl (941) call.

```
SHM_RND = 2 shl 12
```

This constant is used in the shmctl (941) call.

```
SHM_UNLOCK = 12
```

This constant is used in the shmctl (941) call.

```
SHM_W = 2 shl 6
```

This constant is used in the shmctl (941) call.

58.3.2 Types

```
key_t = TKey
```

Alias for TKey (929) type.

```
msglen_t = culong
```

Message length type.

```
msgqnum_t = culong
```

Message queue number type.

```
PIPC_Perm = ^TIPC_Perm
```

Pointer to TIPC_Perm (944) record.

```
PMSG = ^TMSG
```

Pointer to TMSG (944) record.

```
PMSGbuf = ^TMSGbuf
```

Pointer to TMsgBuf (944) record.

```
PMSQid_ds = ^TMSQid_ds
```

Pointer to TMSQid_ds (945).

```
PSEMbuf = ^TSEMbuf
```

Pointer to TSembuf (945) record.

```
PSEMid_ds = ^TSEMid_ds
```

Pointer to TSEMid_ds (946) record.

```
PSEMinfo = ^TSEMinfo
```

Pointer to TSEMinfo (946) record.

```
PSEMun = ^TSEMun
```

Pointer to TSEMun (929) record.

```
PShmid_DS = ^TShmid_ds
```

Pointer to TSHMid_ds (946) record.

```
PSHM_info = ^TSHM_info
```

```
TKey = cint
```

Type returned by the ftok (929) key generating function.

```
TSEMun = record
case cint of
0: (
  val : cint;
);
1: (
  buf : PSEMid_ds
  ;
);
2: (
  arr : PWord;
);
3: (
  padbuf : PSEMinfo;
);
4: (
  padpad
  : pointer;
);
end
```

Record used in semctl (934) call.

58.4 Procedures and functions

58.4.1 ftok

Synopsis: Create token from filename.

Declaration: `function ftok(Path: PChar; ID: cint) : TKey`

Visibility: default

Description: `ftok` returns a key that can be used in a semget (939)shmget (943) or msgget (932) call to access a new or existing IPC resource.

`Path` is the name of a file in the file system, `ID` is a character of your choice. The ftok call does the same as it's C counterpart, so a pascal program and a C program will access the same resource if they use the same `Path` and `ID`

For an example, see msgctl (930), semctl (934) or shmctl (941).

Errors: `ftok` returns -1 if the file in `Path` doesn't exist.

See also: semget (939), shmget (943), msgget (932)

58.4.2 msgctl

Synopsis: Perform various operations on a message queue.

Declaration: `function msgctl(msqid: cint; cmd: cint; buf: PMSQid_ds) : cint`

Visibility: default

Description: `msgctl` performs various operations on the message queue with id `ID`. Which operation is performed, depends on the `cmd` parameter, which can have one of the following values:

IPC_STAT In this case, the `msgctl` call fills the `TMSQid_ds` structure with information about the message queue.

IPC_SETin this case, the `msgctl` call sets the permissions of the queue as specified in the `ipc_perm` record inside `buf`.

IPC_RMID If this is specified, the message queue will be removed from the system.

`buf` contains the data that are needed by the call. It can be `Nil` in case the message queue should be removed.

The function returns `True` if success full, `False` otherwise.

Errors: On error, `False` is returned, and `IPCerror` is set accordingly.

See also: msgget (932), msgsnd (933), msgrcv (933)

Listing: ./examples/ipcex/msgtool.pp

```
program msgtool;

Uses ipc,baseunix;

Type
  PMyMsgBuf = ^TMyMsgBuf;
  TMyMsgBuf = record
    mtype : clong;
    mtext : string[255];
  end;

Procedure DoError (Const Msg : string);

begin
  WriteLn (msg,' returned an error : ',fpgeterrno);
```

```
  halt(1);
end;

Procedure SendMessage (Id : Longint;
                       Var Buf : TMyMsgBuf;
                       MType : Longint;
                       Const MText : String);

begin
  Writeln ('Sending message.');
  Buf.mtype:=mtype;
  Buf.Mtext:=mtext;
  If  msgsnd(Id,PMsgBuf(@Buf),256,0)=-1 then
    DoError('msgsnd');
end;

Procedure ReadMessage (ID : Longint;
                       Var Buf : TMyMsgBuf;
                       MType : longint);

begin
  Writeln ('Reading message.');
  Buf.MType:=MType;
  If msgrcv(ID,PMSGBuf(@Buf),256,mtype,0)<>-1 then
    Writeln ('Type : ',buf.mtype,' Text : ',buf.mtext)
  else
    DoError ('msgrcv');
end;

Procedure RemoveQueue ( ID : Longint);

begin
  If msgctl (id,IPC_RMID,Nil)<>-1 then
    Writeln ('Removed Queue with id ',Id);
end;

Procedure ChangeQueueMode (ID,mode : longint);

Var QueueDS : TMSQid_ds;

begin
  If  msgctl (Id,IPC_STAT,@QueueDS)=-1 then
    DoError ('msgctl : stat');
  Writeln ('Old permissions : ',QueueDS.msg_perm.mode);
  QueueDS.msg_perm.mode:=Mode;
  if msgctl (ID,IPC_SET,@QueueDS)=0 then
    Writeln ('New permissions : ',QueueDS.msg_perm.mode)
  else
   DoError ('msgctl : IPC_SET');
end;

procedure usage;

begin
  Writeln ('Usage : msgtool s(end)    <type> <text> (max 255 characters)');
  Writeln ('                r(eceive) <type>');
  Writeln ('                d(elete)');
  Writeln ('                m(ode) <decimal mode>');
```

```
  halt(1);
end;

Function StrToInt (S : String): longint;

Var M : longint;
    C : Integer;

begin
  val (S,M,C);
  If C<>0 Then DoError ('StrToInt : '+S);
  StrToInt:=M;
end;

Var
  Key : TKey;
  ID  : longint;
  Buf : TMyMsgBuf;

const ipckey = '.'#0;

begin
  If Paramcount<1 then Usage;
  key :=Ftok(@ipckey[1],ord('M'));
  ID:=msgget(key,IPC_CREAT or 438);
  If ID<0 then DoError ('MsgGet');
  Case upCase(Paramstr(1)[1]) of
   'S' : If ParamCount<>3 then
           Usage
         else
           SendMessage (id,Buf,StrToInt(Paramstr(2)),paramstr(3));
   'R' : If ParamCount<>2 then
           Usage
         else
           ReadMessage (id,buf,strtoint(Paramstr(2)));
   'D' : If ParamCount<>1 then
           Usage
         else
           RemoveQueue (ID);
   'M' : If ParamCount<>2 then
           Usage
         else
           ChangeQueueMode (id,strtoint(paramstr(2)));
   else
     Usage
   end;
end.
```

58.4.3 msgget

Synopsis: Return message queue ID, possibly creating the queue.

Declaration: `function msgget(key: TKey; msgflg: cint) : cint`

Visibility: default

Description: `msgget` returns the ID of the message queue described by `key`. Depending on the flags in `msgflg`, a new queue is created.

msgflg can have one or more of the following values (combined by ORs):

IPC_CREAT The queue is created if it doesn't already exist.

IPC_EXCL If used in combination with IPC_CREAT, causes the call to fail if the queue already exists. It cannot be used by itself.

Optionally, the flags can be ORed with a permission mode, which is the same mode that can be used in the file system.

For an example, see msgctl (930).

Errors: On error, -1 is returned, and IPCError is set.

See also: ftok (929), msgsnd (933), msgrcv (933), msgctl (930)

58.4.4 msgrcv

Synopsis: Retrieve a message from the queue.

Declaration:
```
function msgrcv(msqid: cint; msgp: PMSGbuf; msgsz: size_t;
                msgtyp: clong; msgflg: cint) : cint
```

Visibility: default

Description: msgrcv retrieves a message of type msgtyp from the message queue with ID msqid. msgtyp corresponds to the mtype field of the TMSGbuf record. The message is stored in the MSGbuf structure pointed to by msgp.

The msgflg parameter can be used to control the behaviour of the msgrcv call. It consists of an ORed combination of the following flags:

0 No special meaning.

IPC_NOWAIT if no messages are available, then the call returns immediately, with the ENOMSG error.

MSG_NOERROR If the message size is wrong (too large), no error is generated, instead the message is truncated. Normally, in such cases, the call returns an error (E2BIG)

The function returns True if the message was received correctly, False otherwise.

For an example, see msgctl (930).

Errors: In case of error, False is returned, and IPCerror is set.

See also: msgget (932), msgsnd (933), msgctl (930)

58.4.5 msgsnd

Synopsis: Send a message to the message queue.

Declaration:
```
function msgsnd(msqid: cint; msgp: PMSGbuf; msgsz: size_t; msgflg: cint)
                : cint
```

Visibility: default

Description: msgsend sends a message to a message queue with ID msqid. msgp is a pointer to a message buffer, that should be based on the TMsgBuf type. msgsiz is the size of the message (NOT of the message buffer record !)

The msgflg can have a combination of the following values (ORed together):

0No special meaning. The message will be written to the queue. If the queue is full, then the process is blocked.

IPC_NOWAIT If the queue is full, then no message is written, and the call returns immediately.

The function returns `True` if the message was sent successfully, `False` otherwise.

For an example, see msgctl (930).

Errors: In case of error, the call returns `False`, and `IPCerror` is set.

See also: msgget (932), msgrcv (933), msgctl (930)

58.4.6 semctl

Synopsis: Perform various control operations on a semaphore set.

Declaration:
```
function semctl(semid: cint; semnum: cint; cmd: cint; var arg: TSEMun)
                        : cint
```

Visibility: default

Description: `semctl` performs various operations on the semaphore `semnum` w ith semaphore set id `ID`.

The `arg` parameter supplies the data needed for each call. This is a variant record that should be filled differently, according to the command:

```
Type
  TSEMun = record
   case longint of
      0 : ( val : longint );
      1 : ( buf : PSEMid_ds );
      2 : ( arr : PWord );
      3 : ( padbuf : PSeminfo );
      4 : ( padpad : pointer );
   end;
```

Which operation is performed, depends on the `cmd` parameter, which can have one of the following values:

IPC_STAT In this case, the arg record should have it's `buf` field set to the address of a `TSEMid_ds` record. The `semctl` call fills this `TSEMid_ds` structure with information about the semaphore set.

IPC_SET In this case, the `arg` record should have it's `buf` field set to the address of a `TSEMid_ds` record. The `semctl` call sets the permissions of the queue as specified in the `ipc_perm` record.

IPC_RMID If this is specified, the semaphore set is removed from from the system.

GETALLIn this case, the `arr` field of `arg` should point to a memory area where the values of the semaphores will be stored. The size of this memory area is `SizeOf(Word) * Number of semaphores in the set`. This call will then fill the memory array with all the values of the semaphores.

GETNCNTThis will fill the `val` field of the `arg` union with the number of processes waiting for resources.

GETPID`semctl` returns the process ID of the process that performed the last semop (939) call.

GETVAL`semctl` returns the value of the semaphore with number `semnum`.

GETZCNT `semctl` returns the number of processes waiting for semaphores that reach value zero.

SETALL In this case, the `arr` field of `arg` should point to a memory area where the values of the semaphores will be retrieved from. The size of this memory area is `SizeOf(Word) * Number of semaphores in the set`. This call will then set the values of the semaphores from the memory array.

SETVAL This will set the value of semaphore `semnum` to the value in the `val` field of the `arg` parameter.

The function returns -1 on error.

Errors: The function returns -1 on error, and `IPCerror` is set accordingly.

See also: semget (939), semop (939)

Listing: ./examples/ipcex/semtool.pp

```
Program semtool;

{ Program to demonstrate the use of semaphores }

Uses ipc,baseunix;

Const MaxSemValue = 5;

Procedure DoError (Const Msg : String);
var
  error: cint;
begin
  error:=fpgeterrno;
  Writeln ('Error : ',msg,' Code : ',error);
  Halt(1);
end;

Function getsemval (ID,Member : longint) : longint;

Var S : TSEMun;

begin
  GetSemVal:=SemCtl(id,member,SEM_GETVAL,S);
end;

Procedure DispVal (ID,member : longint);

begin
  writeln ('Value for member ',member,' is ',GetSemVal(ID,Member));
end;

Function GetMemberCount (ID : Longint) : longint;

Var opts : TSEMun;
    semds : TSEMid_ds;

begin
  opts.buf:=@semds;
  If semctl(Id,0,IPC_STAT,opts)<>-1 then
    GetMemberCount:=semds.sem_nsems
  else
    GetMemberCount:=-1;
```

```
end;

Function OpenSem (Key : TKey) : Longint;

begin
  OpenSem:=semget(Key,0,438);
  If OpenSem=-1 then
    DoError ('OpenSem');
end;

Function CreateSem (Key : TKey; Members : Longint) : Longint;

Var Count : Longint;
    Semopts : TSemun;

begin
// the semmsl constant seems kernel specific
{  If members>semmsl then
    DoError ('Sorry, maximum number of semaphores in set exceeded');
}
  WriteIn ('Trying to create a new semaphore set with ',members,' members.');
  CreateSem:=semget(key,members,IPC_CREAT or IPC_Excl or 438);
  If CreateSem=-1 then
    DoError ('Semaphore set already exists.');
  Semopts.val:=MaxSemValue; { Initial value of semaphores }
  For Count:=0 to Members-1 do
    semctl(CreateSem,count,SEM_SETVAL,semopts);
end;

Procedure lockSem (ID,Member: Longint);

Var lock : TSEMbuf;

begin
  With lock do
    begin
    sem_num:=0;
    sem_op:=-1;
    sem_flg:=IPC_NOWAIT;
    end;
   if (member<0) or (member>GetMemberCount(ID)-1) then
     DoError ('semaphore member out of range');
   if getsemval(ID,member)=0 then
     DoError ('Semaphore resources exhausted (no lock)');
   lock.sem_num:=member;
   WriteIn ('Attempting to lock member ',member, ' of semaphore ',ID);
   if semop(Id,@lock,1)=-1 then
     DoError ('Lock failed')
   else
     WriteIn ('Semaphore resources decremented by one');
   dispval(ID,Member);
end;

Procedure UnlockSem (ID,Member: Longint);

Var Unlock : TSEMbuf;

begin
```

```
  With Unlock do
    begin
    sem_num:=0;
    sem_op:=1;
    sem_flg:=IPC_NOWAIT;
    end;
   if (member<0) or (member>GetMemberCount(ID)-1) then
     DoError ('semaphore member out of range');
   if getsemval(ID,member)=MaxSemValue then
     DoError ('Semaphore not locked');
   Unlock.sem_num:=member;
   Writeln ('Attempting to unlock member ',member, ' of semaphore ',ID);
   if semop(Id,@unlock,1)=-1 then
     DoError ('Unlock failed')
   else
     Writeln ('Semaphore resources incremented by one');
   dispval(ID,Member);
end;

Procedure RemoveSem (ID : longint);

var S : TSemun;

begin
  If semctl(Id,0,IPC_RMID,s)<>-1 then
    Writeln ('Semaphore removed')
  else
    DoError ('Couldn''t remove semaphore');
end;


Procedure ChangeMode (ID,Mode : longint);

Var rc : longint;
    opts : TSEMun;
    semds : TSEMid_ds;

begin
  opts.buf:=@semds;
  If not semctl (Id,0,IPC_STAT,opts)<>-1 then
    DoError ('Couldn''t stat semaphore');
  Writeln ('Old permissions were : ',semds.sem_perm.mode);
  semds.sem_perm.mode:=mode;
  If semctl(id,0,IPC_SET,opts)<>-1 then
    Writeln ('Set permissions to ',mode)
  else
    DoError ('Couldn''t set permissions');
end;

Procedure PrintSem (ID : longint);

Var I,cnt : longint;

begin
  cnt:=getmembercount(ID);
  Writeln ('Semaphore ',ID,' has ',cnt,' Members');
  For I:=0 to cnt-1 Do
    DispVal(id,i);
```

```
end;

Procedure USage;

begin
  WriteIn ('Usage : semtool c(reate) <count>');
  WriteIn ('                l(ock) <member>');
  WriteIn ('                u(nlock) <member>');
  WriteIn ('                d(elete)');
  WriteIn ('                m(ode) <mode>');
  WriteIn ('                p(rint)');
  halt(1);
end;

Function StrToInt (S : String): longint;

Var M : longint;
    C : Integer;

begin
  val (S,M,C);
  If C<>0 Then DoError ('StrToInt : '+S);
  StrToInt:=M;
end;

Var Key : TKey;
    ID : Longint;


const ipckey='.'#0;

begin
  If ParamCount<1 then USage;
  key:=ftok(@ipckey[1],ORD('s'));
  Case UpCase(Paramstr(1)[1]) of
   'C' : begin
         if paramcount<>2 then usage;
         CreateSem (key,strtoint(paramstr(2)));
         end;
   'L' : begin
         if paramcount<>2 then usage;
         ID:=OpenSem (key);
         LockSem (ID,strtoint(paramstr(2)));
         end;
   'U' : begin
         if paramcount<>2 then usage;
         ID:=OpenSem (key);
         UnLockSem (ID,strtoint(paramstr(2)));
         end;
   'M' : begin
         if paramcount<>2 then usage;
         ID:=OpenSem (key);
         ChangeMode (ID,strtoint(paramstr(2)));
         end;
   'D' : Begin
         ID:=OpenSem(Key);
         RemoveSem(Id);
         end;
```

```
    'P' : begin
          ID:=OpenSem(Key);
          PrintSem(Id);
          end;
  else
    Usage
  end;
end.
```

58.4.7 semget

Synopsis: Return the ID of a semaphore set, possibly creating the set.

Declaration: `function semget(key: TKey; nsems: cint; semflg: cint) : cint`

Visibility: default

Description: `msgget` returns the ID of the semaphore set described by `key`. Depending on the flags in `semflg`, a new queue is created.

`semflg` can have one or more of the following values (combined by ORs):

IPC_CREAT The queue is created if it doesn't already exist.

IPC_EXCL If used in combination with `IPC_CREAT`, causes the call to fail if the set already exists. It cannot be used by itself.

Optionally, the flags can be `OR`ed with a permission mode, which is the same mode that can be used in the file system.

if a new set of semaphores is created, then there will be `nsems` semaphores in it.

Errors: On error, -1 is returned, and `IPCError` is set.

See also: ftok (929), semop (939), semctl (934)

58.4.8 semop

Synopsis: Perform semaphore operation.

Declaration: `function semop(semid: cint; sops: PSEMbuf; nsops: cuint) : cint`

Visibility: default

Description: `semop` performs a set of operations on a message queue. `sops` points to an array of type `TSEMbuf`. The array should contain `nsops` elements.

The fields of the TSEMbuf (945) structure

```
TSEMbuf = record
    sem_num : word;
    sem_op  : integer;
    sem_flg : integer;
```

should be filled as follows:

sem_num The number of the semaphore in the set on which the operation must be performed.

sem_op The operation to be performed. The operation depends on the sign of `sem_op`: A positive number is simply added to the current value of the semaphore. If 0 (zero) is specified, then the process is suspended until the specified semaphore reaches zero. If a negative number is specified, it is subtracted from the current value of the semaphore. If the value would become negative then the process is suspended until the value becomes big enough, unless `IPC_NOWAIT` is specified in the `sem_flg`.

sem_flg Optional flags: if `IPC_NOWAIT` is specified, then the calling process will never be suspended.

The function returns `True` if the operations were successful, `False` otherwise.

Errors: In case of error, `False` is returned, and `IPCerror` is set.

See also: semget (939), semctl (934)

58.4.9 semtimedop

Synopsis: Perform semaphore operation using timeout.

Declaration:
```
function semtimedop(semid: cint; sops: PSEMbuf; nsops: cuint;
                    timeOut: ptimespec) : cint
```

Visibility: default

Description: `semop` performs a set of operations on a message queue, just as semop (939). `sops` points to an array of type `TSEMbuf`. The array should contain `nsops` elements. The `timeOut` argument points to a time specification: if the operations cannot be performed withing the specified, time, the function will return with an error.

For more information on the actual operations, see semop (939).

See also: semop (939)

58.4.10 shmat

Synopsis: Attach a shared memory block.

Declaration:
```
function shmat(shmid: cint; shmaddr: pointer; shmflg: cint) : pointer
```

Visibility: default

Description: `shmat` attaches a shared memory block with identified `shmid` to the current process. The function returns a pointer to the shared memory block.

If `shmaddr` is `Nil`, then the system chooses a free unmapped memory region, as high up in memory space as possible.

If `shmaddr` is non-nil, and `SHM_RND` is in `shmflg`, then the returned address is `shmaddr`, rounded down to `SHMLBA`. If `SHM_RND` is not specified, then `shmaddr` must be a page-aligned address.

The parameter `shmflg` can be used to control the behaviour of the `shmat` call. It consists of a ORed combination of the following constants:

SHM_RND The suggested address in `shmaddr` is rounded down to `SHMLBA`.

SHM_RDONLYthe shared memory is attached for read access only. Otherwise the memory is attached for read-write. The process then needs read-write permissions to access the shared memory.

For an example, see shmctl (941).

Errors: If an error occurs, -1 is returned, and `IPCerror` is set.

See also: shmget (943), shmdt (943), shmctl (941)

58.4.11 shmctl

Synopsis: Perform control operations on a shared memory block.

Declaration: `function shmctl(shmid: cint; cmd: cint; buf: PShmid_DS) : cint`

Visibility: default

Description: `shmctl` performs various operations on the shared memory block identified by identifier `shmid`.

The `buf` parameter points to a `TSHMid_ds` record. The `cmd` parameter is used to pass which operation is to be performed. It can have one of the following values :

IPC_STAT `shmctl` fills the `TSHMid_ds` record that `buf` points to with the available information about the shared memory block.

IPC_SET applies the values in the `ipc_perm` record that `buf` points to, to the shared memory block.

IPC_RMID the shared memory block is destroyed (after all processes to which the block is attached, have detached from it).

If successful, the function returns `True`, `False` otherwise.

Errors: If an error occurs, the function returns `False`, and `IPCerror` is set.

See also: shmget (943), shmat (940), shmdt (943)

Listing: ./examples/ipcex/shmtool.pp

```
Program shmtool;

uses ipc,strings,Baseunix;

Const SegSize = 100;

var key : Tkey;
    shmid,cntr : longint;
    segptr : pchar;

Procedure USage;

begin
 Writeln ('Usage : shmtool w(rite) text');
 writeln ('                r(ead)');
 writeln ('                d(elete)');
 writeln ('                m(ode change) mode');
 halt(1);
end;

Procedure Writeshm (ID : Longint; ptr : pchar; S : string);

begin
  strpcopy (ptr,s);
```

```
end;

Procedure Readshm(ID : longint; ptr : pchar);

begin
  Writeln ('Read : ',ptr);
end;

Procedure removeshm (ID : Longint);

begin
  shmctl (ID,IPC_RMID,Nil);
  writeln ('Shared memory marked for deletion');
end;

Procedure CHangeMode (ID : longint; mode : String);

Var m : word;
    code : integer;
    data : TSHMid_ds;

begin
  val (mode,m,code);
  if code<>0 then
    usage;
  If shmctl (shmid,IPC_STAT,@data)=-1 then
    begin
     writeln ('Error : shmctl :',fpgeterrno);
     halt(1);
    end;
  writeln ('Old permissions : ',data.shm_perm.mode);
  data.shm_perm.mode:=m;
  If shmctl (shmid,IPC_SET,@data)=-1 then
    begin
    writeln ('Error : shmctl :',fpgeterrno);
    halt(1);
    end;
  writeln ('New permissions : ',data.shm_perm.mode);
end;

const ftokpath = '.'#0;

begin
  if paramcount<1 then usage;
   key := ftok (pchar(@ftokpath[1]),ord('S'));
  shmid := shmget(key,segsize,IPC_CREAT or IPC_EXCL or 438);
  If shmid=-1 then
    begin
    Writeln ('Shared memory exists. Opening as client');
    shmid := shmget(key,segsize,0);
    If shmid = -1 then
      begin
      Writeln ('shmget : Error !',fpgeterrno);
      halt(1);
      end
    end
  else
    Writeln ('Creating new shared memory segment.');
```

```
  segptr:=shmat(shmid,nil,0);
  if longint(segptr)=-1 then
    begin
    WriteLn ('Shmat : error !',fpgeterrno);
    halt(1);
    end;
  case upcase(paramstr(1)[1]) of
    'W' : writeshm (shmid,segptr,paramstr(2));
    'R' : readshm (shmid,segptr);
    'D' : removeshm(shmid);
    'M' : changemode (shmid,paramstr(2));
  else
    begin
    writeln (paramstr(1));
    usage;
    end;
  end;
end.
```

58.4.12 shmdt

Synopsis: Detach shared memory block.

Declaration: `function shmdt(shmaddr: pointer) : cint`

Visibility: default

Description: `shmdt` detaches the shared memory at address `shmaddr`. This shared memory block is unavailable to the current process, until it is attached again by a call to shmat (940).

The function returns `True` if the memory block was detached successfully, `False` otherwise.

Errors: On error, False is returned, and IPCerror is set.

See also: shmget (943), shmat (940), shmctl (941)

58.4.13 shmget

Synopsis: Return the ID of a shared memory block, possibly creating it.

Declaration: `function shmget(key: TKey; size: size_t; flag: cint) : cint`

Visibility: default

Description: `shmget` returns the ID of a shared memory block, described by `key`. Depending on the flags in `flag`, a new memory block is created.

`flag` can have one or more of the following values (combined by ORs):

IPC_CREAT The queue is created if it doesn't already exist.

IPC_EXCL If used in combination with `IPC_CREAT`, causes the call to fail if the queue already exists. It cannot be used by itself.

Optionally, the flags can be `OR`ed with a permission mode, which is the same mode that can be used in the file system.

if a new memory block is created, then it will have size `Size` bytes in it.

Errors: On error, -1 is returned, and `IPCError` is set.

58.5 TIPC_Perm

```
TIPC_Perm = record
  key : TKey;
  uid : kernel_uid_t;
  gid : kernel_gid_t
  ;
  cuid : kernel_uid_t;
  cgid : kernel_gid_t;
  mode : kernel_mode_t
  ;
  __pad1 : Array[1..4-sizeof(mode_t)] of Byte;
  seq : cushort;
  __pad2 : cushort;
  __unused1 : culong;
  __unused2 : culong;
end
```

`TIPC_Perm` is used in all IPC systems to specify the permissions. It should never be used directly.

58.6 TMSG

```
TMSG = record
  msg_next : PMSG;
  msg_type : LongInt;
  msg_spot
  : PChar;
  msg_stime : LongInt;
  msg_ts : Integer;
end
```

Record used in the handling of message queues. Do not use directly.

58.7 TMSGbuf

```
TMSGbuf = record
  mtype : clong;
  mtext : Array[0..0] of Char;
end
```

The `TMSGbuf` record is a record containing the data of a record. you should never use this record directly, instead you should make your own record that follows the structure of the `TMSGbuf` record, but that has a size that is big enough to accommodate your messages. The `mtype` field should always be present, and should always be filled.

58.8 TMSGinfo

```
TMSGinfo = record
```

```
  msgpool : cint;
  msgmap : cint;
  msgmax : cint
  ;
  msgmnb : cint;
  msgmni : cint;
  msgssz : cint;
  msgtql : cint
  ;
  msgseg : cushort;
end
```

Internal message system record. Do not use directly.

58.9 TMSQid_ds

```
TMSQid_ds = record
  msg_perm : TIPC_Perm;
  msg_stime : time_t;
  msg_rtime : time_t;
  msg_ctime : time_t;
  msg_cbytes : QWord;
  msg_qnum : QWord;
  msg_qbytes : QWord;
  msg_lspid : ipc_pid_t
  ;
  msg_lrpid : ipc_pid_t;
  pad1 : QWord;
  pad2 : QWord;
end
```

This record should never be used directly, it is an internal kernel record. It's fields may change at any time.

58.10 TSEMbuf

```
TSEMbuf = record
  sem_num : cushort;
  sem_op : cshort;
  sem_flg
  : cshort;
end
```

The `TSEMbuf` record is used in the semop (939) call, and is used to specify which operations you want to do.

58.11 TSEMid_ds

```
TSEMid_ds = record
```

```
  sem_perm : TIPC_Perm;
  sem_otime : time_t;
  unused1 : culong;
  sem_ctime : time_t;
  unused2 : culong;
  sem_nsems
  : culong;
  unused3 : culong;
  unused4 : culong;
end
```

Structure returned by the semctl (934) call, contains all data of a semaphore.

58.12 TSEMinfo

```
TSEMinfo = record
  semmap : cint;
  semmni : cint;
  semmns : cint
  ;
  semmnu : cint;
  semmsl : cint;
  semopm : cint;
  semume : cint
  ;
  semusz : cint;
  semvmx : cint;
  semaem : cint;
end
```

Internal semaphore system record. Do not use.

58.13 TShmid_ds

```
TShmid_ds = record
  shm_perm : TIPC_Perm;
  shm_segsz : size_t;
  shm_atime : time_t;
  shm_dtime : time_t;
  shm_ctime : time_t;
  shm_cpid : pid_t;
  shm_lpid : pid_t;
  shm_nattch : culong;
  __unused4 : culong;
  __unused5 : culong;
end
```

Record used in the shmctl (941) call to set or retrieve settings for shared memory.

58.14 TSHMinfo

```
TSHMinfo = record
  shmmax : cint;
  shmmin : cint;
  shmmni : cint
  ;
  shmseg : cint;
  shmall : cint;
end
```

Record used by the shared memory system, Do not use directly.

58.15 TSHM_info

```
TSHM_info = record
  used_ids : cint;
  shm_tot : culong;
  shm_rss
  : culong;
  shm_swp : culong;
  swap_attempts : culong;
  swap_successes
  : culong;
end
```

Chapter 59

Reference for unit 'keyboard'

59.1 Used units

Table 59.1: Used units by unit 'keyboard'

Name	Page
System	1362

59.2 Overview

The Keyboard unit implements a keyboard access layer which is system independent. It can be used to poll the keyboard state and wait for certain events. Waiting for a keyboard event can be done with the GetKeyEvent (962) function, which will return a driver-dependent key event. This key event can be translated to a interpretable event by the TranslateKeyEvent (970) function. The result of this function can be used in the other event examining functions.

A custom keyboard driver can be installed using the SetKeyboardDriver (969) function. The current keyboard driver can be retrieved using the GetKeyboardDriver (961) function. The last section of this chapter demonstrates how to make a keyboard driver.

59.3 Unix specific notes.

On Unix, applications run on a "terminal", and the application writes to the screen and reads from the keyboard by communicating with the terminal. Unix keyboard handling is mostly backward compatible with the DEC VT100 and VT220 terminals from tens of years ago. The VT100 and VT220 had very different keyboards than todays PC's and this is where the problems start. To make it worse the protocol of both terminals has not been very well designed.

Because of this, the keyboard unit on Unix operating systems does a best effort to provide keyboard functionality. An implementation with full keyboard facilities like on other operating systems is not possible.

The exception is the Linux kernel. The terminal emulation of the Linux kernel is from a PC keyboard viewpoint hopeless as well, but unlike other terminal emulators it is configurable. On the Linux console, the Free Pascal keyboard unit tries to implement full functionality.

Users of applications using the keyboard unit should expect the following:

- Full functionality on the Linux console. It must be the bare console, SSH into another machine will kill the full functionality.
- Limited functionality otherwise.

Notes about Linux full functionality:

- The keyboard is reprogrammed. If the keyboard is for whatever reason not restored in its original state, please load your keymap to reinitialize it.
- Alt+function keys generate keycodes for those keys. To switch virtual consoles, use ctrl+alt+function key.
- Unlike what you're used to with other Unix software, escape works as you intuitively expect, it generates the keycode for an escape key **without a delay.**

The limited functionality does include these quirks:

- Escape must be pressed two times before it has effect.
- On the Linux console, when the users runs the program by logging into another machine:
  - Shift+F1 and Shift+F12 will generate keycodes for F11 and F12.
  - Shift+arrow keys, shift+ins, shift+del, shift+home, shift+end do not work. The same is true about the control and alt combinations.
  - Alt+function keys will switch virtual consoles instead of generating the right key sequences.
  - Ctrl+function keys will generate the keycodes for the function keys without ctrl
- In Xterm:
  - Shift+insert pastes the x clipboard, no keycode will be generated.
- In Konsole:
  - Shift+insert pastes the x clipboard, no keycode will be generated.
  - Shift+arrow keys doesn't work, nor does ctrl+arrow keys

If you have a non-standard terminal, some keys may not work at all. When in limited functionality mode, the user can work around using an escape prefix:

- Esc+1 = F1, Esc+2 = F2.
- Esc before another key is equal to alt+key.

In such cases, if the terminal does output an escape sequence for those keys, please submit a bug report so we can add them.

59.4 Writing a keyboard driver.

Writing a keyboard driver means that hooks must be created for most of the keyboard unit functions. The `TKeyBoardDriver` record contains a field for each of the possible hooks:

```
TKeyboardDriver = Record
  InitDriver : Procedure;
  DoneDriver : Procedure;
  GetKeyEvent : Function : TKeyEvent;
  PollKeyEvent : Function : TKeyEvent;
  GetShiftState : Function : Byte;
  TranslateKeyEvent : Function (KeyEvent: TKeyEvent): TKeyEvent;
  TranslateKeyEventUniCode: Function (KeyEvent: TKeyEvent): TKeyEvent;
end;
```

The meaning of these hooks is explained below:

InitDriver Called to initialize and enable the driver. Guaranteed to be called only once. This should initialize all needed things for the driver.

DoneDriver Called to disable and clean up the driver. Guaranteed to be called after a call to `initDriver`. This should clean up all things initialized by `InitDriver`.

GetKeyEvent Called by GetKeyEvent (962). Must wait for and return the next key event. It should NOT store keys.

PollKeyEvent Called by PollKeyEvent (967). It must return the next key event if there is one. Should not store keys.

GetShiftState Called by PollShiftStateEvent (967). Must return the current shift state.

TranslateKeyEvent Should translate a raw key event to a correct key event, i.e. should fill in the shiftstate and convert function key scancodes to function key keycodes. If the `TranslateKeyEvent` is not filled in, a default translation function will be called which converts the known scancodes from the tables in the previous section to a correct keyevent.

TranslateKeyEventUniCode Should translate a key event to a UNICode key representation.

Strictly speaking, only the `GetKeyEvent` and `PollKeyEvent` hooks must be implemented for the driver to function correctly.

The example unit demonstrates how a keyboard driver can be installed. It takes the installed driver, and hooks into the `GetKeyEvent` function to register and log the key events in a file. This driver can work on top of any other driver, as long as it is inserted in the `uses` clause *after* the real driver unit, and the real driver unit should set the driver record in its initialization section.

Note that with a simple extension of this unit could be used to make a driver that is capable of recording and storing a set of keyboard strokes, and replaying them at a later time, so a 'keyboard macro' capable driver. This driver could sit on top of any other driver.

Listing: ./examples/kbdex/logkeys.pp

```
unit logkeys;

interface

Procedure StartKeyLogging;
Procedure StopKeyLogging;
```

```
Function  IsKeyLogging : Boolean;
Procedure  SetKeyLogFileName(FileName : String);


implementation

uses sysutils,keyboard;

var
  NewKeyBoardDriver,
  OldKeyBoardDriver : TKeyboardDriver;
  Active,Logging : Boolean;
  LogFileName : String;
  KeyLog : Text;

Function TimeStamp : String;

begin
  TimeStamp:=FormatDateTime('hh:nn:ss',Time());
end;

Procedure StartKeyLogging;

begin
  Logging:=True;
  Writeln(KeyLog,'Start logging keystrokes at: ',TimeStamp);
end;

Procedure StopKeyLogging;

begin
  Writeln(KeyLog,'Stop logging keystrokes at: ',TimeStamp);
  Logging:=False;
end;

Function IsKeyLogging : Boolean;

begin
  IsKeyLogging:=Logging;
end;

Function LogGetKeyEvent : TKeyEvent;

Var
  K : TKeyEvent;

begin
  K:=OldkeyboardDriver.GetKeyEvent();
  If Logging then
    begin
    Write(KeyLog,TimeStamp,': Key event: ');
    Writeln(KeyLog,KeyEventToString(TranslateKeyEvent(K)));
    end;
  LogGetKeyEvent:=K;
end;

Procedure LogInitKeyBoard;
```

```
begin
  OldKeyBoardDriver.InitDriver();
  Assign(KeyLog,logFileName);
  Rewrite(KeyLog);
  Active:=True;
  StartKeyLogging;
end;

Procedure LogDoneKeyBoard;

begin
  StopKeyLogging;
  Close(KeyLog);
  Active:=False;
  OldKeyBoardDriver.DoneDriver();
end;

Procedure SetKeyLogFileName(FileName : String);

begin
  If Not Active then
    LogFileName:=FileName;
end;

Initialization
  GetKeyBoardDriver(OldKeyBoardDriver);
  NewKeyBoardDriver:=OldKeyBoardDriver;
  NewKeyBoardDriver.GetKeyEvent:=@LogGetKeyEvent;
  NewKeyBoardDriver.InitDriver:=@LogInitKeyboard;
  NewKeyBoardDriver.DoneDriver:=@LogDoneKeyboard;
  LogFileName:='keyboard.log';
  Logging:=False;
  SetKeyboardDriver(NewKeyBoardDriver);
end.
```

Listing: ./examples/kbdex/ex9.pp

```
program example9;

{ This program demonstrates the logkeys unit }

uses keyboard,logkeys;

Var
  K : TKeyEvent;

begin
  InitKeyBoard;
  Writeln('Press keys, press "q" to end, "s" toggles logging.');
  Repeat
    K:=GetKeyEvent;
    K:=TranslateKeyEvent(K);
    Writeln('Got key : ',KeyEventToString(K));
    if GetKeyEventChar(K)='s' then
      if IsKeyLogging then
        StopKeyLogging
      else
        StartKeyLogging;
  Until (GetKeyEventChar(K)='q');
```

```
  DoneKeyBoard;
end.
```

59.5 Keyboard scan codes.

Special physical keys are encoded with the DOS scan codes for these keys in the second byte of the TKeyEvent (960) type. A complete list of scan codes can be found in the below table. This is the list of keys that is used by the default key event translation mechanism. When writing a keyboard driver, either these constants should be returned by the various key event functions, or the `TranslateKeyEvent` hook should be implemented by the driver.

Table 59.2: Key Scancodes

Code	Key	Code	Key	Code	Key
00	NoKey	3D	F3	70	ALT-F9
01	ALT-Esc	3E	F4	71	ALT-F10
02	ALT-Space	3F	F5	72	CTRL-PrtSc
04	CTRL-Ins	40	F6	73	CTRL-Left
05	SHIFT-Ins	41	F7	74	CTRL-Right
06	CTRL-Del	42	F8	75	CTRL-end
07	SHIFT-Del	43	F9	76	CTRL-PgDn
08	ALT-Back	44	F10	77	CTRL-Home
09	ALT-SHIFT-Back	47	Home	78	ALT-1
0F	SHIFT-Tab	48	Up	79	ALT-2
10	ALT-Q	49	PgUp	7A	ALT-3
11	ALT-W	4B	Left	7B	ALT-4
12	ALT-E	4C	Center	7C	ALT-5
13	ALT-R	4D	Right	7D	ALT-6
14	ALT-T	4E	ALT-GrayPlus	7E	ALT-7
15	ALT-Y	4F	end	7F	ALT-8
16	ALT-U	50	Down	80	ALT-9
17	ALT-I	51	PgDn	81	ALT-0
18	ALT-O	52	Ins	82	ALT-Minus
19	ALT-P	53	Del	83	ALT-Equal
1A	ALT-LftBrack	54	SHIFT-F1	84	CTRL-PgUp
1B	ALT-RgtBrack	55	SHIFT-F2	85	F11
1E	ALT-A	56	SHIFT-F3	86	F12
1F	ALT-S	57	SHIFT-F4	87	SHIFT-F11
20	ALT-D	58	SHIFT-F5	88	SHIFT-F12
21	ALT-F	59	SHIFT-F6	89	CTRL-F11
22	ALT-G	5A	SHIFT-F7	8A	CTRL-F12
23	ALT-H	5B	SHIFT-F8	8B	ALT-F11
24	ALT-J	5C	SHIFT-F9	8C	ALT-F12
25	ALT-K	5D	SHIFT-F10	8D	CTRL-Up
26	ALT-L	5E	CTRL-F1	8E	CTRL-Minus
27	ALT-SemiCol	5F	CTRL-F2	8F	CTRL-Center
28	ALT-Quote	60	CTRL-F3	90	CTRL-GreyPlus
29	ALT-OpQuote	61	CTRL-F4	91	CTRL-Down
2B	ALT-BkSlash	62	CTRL-F5	94	CTRL-Tab
2C	ALT-Z	63	CTRL-F6	97	ALT-Home
2D	ALT-X	64	CTRL-F7	98	ALT-Up
2E	ALT-C	65	CTRL-F8	99	ALT-PgUp
2F	ALT-V	66	CTRL-F9	9B	ALT-Left
30	ALT-B	67	CTRL-F10	9D	ALT-Right
31	ALT-N	68	ALT-F1	9F	ALT-end
32	ALT-M	69	ALT-F2	A0	ALT-Down
33	ALT-Comma	6A	ALT-F3	A1	ALT-PgDn
34	ALT-Period	6B	ALT-F4	A2	ALT-Ins
35	ALT-Slash	6C	ALT-F5	A3	ALT-Del
37	ALT-GreyAst	6D	ALT-F6	A5	ALT-Tab
3B	F1	6E	ALT-F7		
3C	F2	6F	ALT-F8		

A list of scan codes for special keys and combinations with the SHIFT, ALT and CTRL keys can be found in the following table: They are for quick reference only.

Table 59.3: Special keys scan codes

Key	Code	SHIFT-Key	CTRL-Key	Alt-Key
NoKey	00			
F1	3B	54	5E	68
F2	3C	55	5F	69
F3	3D	56	60	6A
F4	3E	57	61	6B
F5	3F	58	62	6C
F6	40	59	63	6D
F7	41	5A	64	6E
F8	42	5B	65	6F
F9	43	5C	66	70
F10	44	5D	67	71
F11	85	87	89	8B
F12	86	88	8A	8C
Home	47		77	97
Up	48		8D	98
PgUp	49		84	99
Left	4B		73	9B
Center	4C		8F	
Right	4D		74	9D
end	4F		75	9F
Down	50		91	A0
PgDn	51		76	A1
Ins	52	05	04	A2
Del	53	07	06	A3
Tab	8	0F	94	A5
GreyPlus			90	4E

59.6 Constants, types and variables

59.6.1 Constants

```
AltPrefix : Byte = 0
```

Keycode for alternate prefix key for Alt key. Unix Only.

```
CtrlPrefix : Byte = 0
```

Keycode for alternate prefix key for Ctrl key. Unix only.

```
errKbdBase = 1010
```

Base of keyboard routine error reporting constants.

```
errKbdInitError = errKbdBase + 0
```

Failed to initialize keyboard driver.

```
errKbdNotImplemented = errKbdBase + 1
```

Keyboard driver not implemented.

```
kbAlt = 8
```

Alt key modifier.

```
kbASCII = $00
```

ASCII code key event.

```
kbCtrl = 4
```

Control key modifier.

```
kbdApps = $FF17
```

Application key (popup-menu) pressed.

```
kbdDelete = $FF2A
```

Delete key pressed.

```
kbdDown = $FF27
```

Arrow down key pressed.

```
kbdEnd = $FF26
```

End key pressed.

```
kbdF1 = $FF01
```

F1 function key pressed.

```
kbdF10 = $FF0A
```

F10 function key pressed.

```
kbdF11 = $FF0B
```

F12 function key pressed.

```
kbdF12 = $FF0C
```

F12 function key pressed.

```
kbdF13 = $FF0D
```

F13 function key pressed.

```
kbdF14 = $FF0E
```

F14 function key pressed.

```
kbdF15 = $FF0F
```

F15 function key pressed.

```
kbdF16 = $FF10
```

F16 function key pressed.

```
kbdF17 = $FF11
```

F17 function key pressed.

```
kbdF18 = $FF12
```

F18 function key pressed.

```
kbdF19 = $FF13
```

F19 function key pressed.

```
kbdF2 = $FF02
```

F2 function key pressed.

```
kbdF20 = $FF14
```

F20 function key pressed.

```
kbdF3 = $FF03
```

F3 function key pressed.

```
kbdF4 = $FF04
```

F4 function key pressed.

```
kbdF5 = $FF05
```

F5 function key pressed.

```
kbdF6 = $FF06
```

F6 function key pressed.

```
kbdF7 = $FF07
```

F7 function key pressed.

```
kbdF8 = $FF08
```

F8 function key pressed.

```
kbdF9 = $FF09
```

F9 function key pressed.

```
kbdHome = $FF20
```

Home key pressed.

```
kbdInsert = $FF29
```

Insert key pressed.

```
kbdLeft = $FF23
```

Arrow left key pressed.

```
kbdLWin = $FF15
```

Left windows key pressed.

```
kbdMiddle = $FF24
```

Middle key pad key pressed (numerical 5).

```
kbdPgDn = $FF28
```

Page down key pressed.

```
kbdPgUp = $FF22
```

Page Up key pressed.

```
kbdRight = $FF25
```

Arrow right key pressed.

```
kbdRWin = $FF16
```

Right windows key pressed.

```
kbdUp = $FF21
```

Arrow up key pressed.

```
kbFnKey = $02
```

function key pressed.

```
kbLeftShift = 1
```

Left shift key modifier.

```
kbPhys = $03
```

Physical key code event.

```
kbReleased = $04
```

Key release event (not implemented in FPC).

```
kbRightShift = 2
```

Right shift key modifier.

```
kbShift = kbLeftShift or kbRightShift
```

Shift key modifier.

```
kbUniCode = $01
```

Unicode code key event.

```
SAnd : string = 'AND'
```

This constant is used as the 'And' word in key descriptions. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

```
ShiftPrefix : Byte = 0
```

Keycode for alternate prefix key for Shift key. Unix Only.

```
SKeyPad : Array[0..($FF2F-kbdHome)] of string = ('Home', 'Up', 'PgUp'
  , 'Left', 'Middle', 'Right', 'End', 'Down', 'PgDn', 'Insert', 'Delete'
  , '', '', '', '', '')
```

This constant describes all keypad keys. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

```
SLeftRight : Array[1..2] of string = ('LEFT', 'RIGHT')
```

This constant contains strings to describe left and right keys. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

```
SScanCode : string = 'Key with scancode '
```

This constant contains a string to denote a scancode key event. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

```
SShift : Array[1..3] of string = ('SHIFT', 'CTRL', 'ALT')
```

This constant describes the various modifier keys. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

```
SUnicodeChar : string = 'Unicode character '
```

This constant contains a string to denote a Unicode key event. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

```
SUnknownFunctionKey : string = 'Unknown function key : '
```

This constant contains a string to denote that an unknown function key was found. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

59.6.2 Types

```
PTreeElement = ^TTreeElement
```

Pointer to TTreeElement (972) record.

```
TKeyEvent = Cardinal
```

The `TKeyEvent` type is the base type for all keyboard events.

The key stroke is encoded in the 4 bytes of the `TKeyEvent` type. The various fields of the key stroke encoding can be obtained by typecasting the `TKeyEvent` type to the TKeyRecord (971) type.

```
Tprocedure = procedure
```

Procedure prototype.

59.7 Procedures and functions

59.7.1 DoneKeyboard

Synopsis: Deactivate keyboard driver.

Declaration: `procedure DoneKeyboard`

Visibility: default

Description: `DoneKeyboard` de-initializes the keyboard interface if the keyboard driver is active. If the keyboard driver is not active, the function does nothing.

This will cause the keyboard driver to clear up any allocated memory, or restores the console or terminal the program was running in to its initial state before the call to InitKeyBoard (965). This function should be called on program exit. Failing to do so may leave the terminal or console window in an unusable state. Its exact action depends on the platform on which the program is running.

On Unix the default keyboard driver restores the line ending of system.output to #10.

For an example, see most other functions.

Errors: None.

See also: InitKeyBoard (965)

59.7.2 FunctionKeyName

Synopsis: Return string representation of a function key code.

Declaration: `function FunctionKeyName(KeyCode: Word) : string`

Visibility: default

Description: `FunctionKeyName` returns a string representation of the function key with code `KeyCode`. This can be an actual function key, or one of the cursor movement keys.

Errors: In case `KeyCode` does not contain a function code, the `SUnknownFunctionKey` string is returned, appended with the `KeyCode`.

See also: ShiftStateToString (969), KeyEventToString (966)

Listing: ./examples/kbdex/ex8.pp

```
Program Example8;

{ Program to demonstrate the FunctionKeyName function. }

Uses keyboard;

Var
  K : TkeyEvent;

begin
  InitKeyboard;
  Writeln('Press function keys, press "q" to end.');
  Repeat
    K:=GetKeyEvent;
    K:=TranslateKeyEvent(K);
    If IsFunctionKey(k) then
      begin
      Write('Got function key : ');
      Writeln(FunctionKeyName(TkeyRecord(K).KeyCode));
      end;
  Until (GetKeyEventChar(K)='q');
  DoneKeyboard;
end.
```

59.7.3 GetKeyboardDriver

Synopsis: Return the current keyboard driver record.

Declaration: `procedure GetKeyboardDriver(var Driver: TKeyboardDriver)`

Visibility: default

Description: `GetKeyBoardDriver` returns in `Driver` the currently active keyboard driver. This function can be used to enhance an existing keyboarddriver.

For more information on getting and setting the keyboard driver kbddriver (950).

Errors: None.

See also: SetKeyboardDriver (969)

59.7.4 GetKeyEvent

Synopsis: Get the next raw key event, wait if needed.

Declaration: `function GetKeyEvent : TKeyEvent`

Visibility: default

Description: `GetKeyEvent` returns the last keyevent if it is available, or waits for one if none is available. A non-blocking version is available in PollKeyEvent (967).

The returned key is encoded as a `TKeyEvent` type variable, and is normally the physical key scan code, (the scan code is driver dependent) which can be translated with one of the translation functions TranslateKeyEvent (970) or TranslateKeyEventUniCode (970). See the types section for a description of how the key is described.

Errors: If no key became available (e.g. when the driver does not support it), 0 is returned.

See also: PutKeyEvent (968), PollKeyEvent (967), TranslateKeyEvent (970), TranslateKeyEventUniCode (970)

Listing: ./examples/kbdex/ex1.pp

```
program example1;

{ This program demonstrates the GetKeyEvent function }

uses keyboard;

Var
  K : TKeyEvent;

begin
  InitKeyBoard;
  Writeln('Press keys, press "q" to end.');
  Repeat
    K:=GetKeyEvent;
    K:=TranslateKeyEvent(K);
    Write('Got key event with ');
    Case GetKeyEventFlags(K) of
      kbASCII     : Writeln('ASCII key');
      kbUniCode   : Writeln('Unicode key');
      kbFnKey     : Writeln('Function key');
      kbPhys      : Writeln('Physical key');
      kbReleased  : Writeln('Released key event');
    end;
    Writeln('Got key : ',KeyEventToString(K));
  Until (GetKeyEventChar(K)='q');
  DoneKeyBoard;
end.
```

59.7.5 GetKeyEventChar

Synopsis: Get the character key part of a key event.

Declaration: `function GetKeyEventChar(KeyEvent: TKeyEvent) : Char`

Visibility: default

Description: `GetKeyEventChar` returns the charcode part of the given `KeyEvent`, if it contains a translated character key keycode. The charcode is simply the ascii code of the character key that was pressed.

It returns the null character if the key was not a character key, but e.g. a function key.

For an example, see GetKeyEvent (962)

Errors: None.

See also: GetKeyEventUniCode (965), GetKeyEventShiftState (964), GetKeyEventFlags (964), GetKeyEventCode (963), GetKeyEvent (962)

59.7.6 GetKeyEventCode

Synopsis: Translate function key part of a key event code.

Declaration: `function GetKeyEventCode(KeyEvent: TKeyEvent) : Word`

Visibility: default

Description: `GetKeyEventCode` returns the translated function keycode part of the given KeyEvent, if it contains a translated function key.

If the key pressed was not a function key, the null character is returned.

Errors: None.

See also: GetKeyEventUniCode (965), GetKeyEventShiftState (964), GetKeyEventFlags (964), GetKeyEventChar (962), GetKeyEvent (962)

Listing: ./examples/kbdex/ex2.pp

```
Program Example2;

{ Program to demonstrate the GetKeyEventCode function. }

Uses keyboard;

Var
  K : TKeyEvent;

begin
  InitKeyBoard;
  Writeln('Press function keys, or press "q" to end.');
  Repeat
    K:=GetKeyEvent;
    K:=TranslateKeyEvent(K);
    If (GetKeyEventFlags(K)<>KbfnKey) then
      Writeln('Not a function key')
    else
      begin
      Write('Got key (',GetKeyEventCode(K));
      Writeln(') : ',KeyEventToString(K));
      end;
  Until (GetKeyEventChar(K)='q');
  DoneKeyboard;
end.
```

59.7.7 GetKeyEventFlags

Synopsis: Extract the flags from a key event.

Declaration: `function GetKeyEventFlags(KeyEvent: TKeyEvent) : Byte`

Visibility: default

Description: `GetKeyEventFlags` returns the flags part of the given `KeyEvent`.

For an example, see GetKeyEvent (962)

Errors: None.

See also: GetKeyEventUniCode (965), GetKeyEventShiftState (964), GetKeyEventCode (963), GetKeyEventChar (962), GetKeyEvent (962)

59.7.8 GetKeyEventShiftState

Synopsis: Return the current state of the shift keys.

Declaration: `function GetKeyEventShiftState(KeyEvent: TKeyEvent) : Byte`

Visibility: default

Description: `GetKeyEventShiftState` returns the shift-state values of the given `KeyEvent`. This can be used to detect which of the modifier keys `Shift`, `Alt` or `Ctrl` were pressed. If none were pressed, zero is returned.

Note that this function does not always return expected results; In a UNIX X-Term, the modifier keys do not always work.

Errors: None.

See also: GetKeyEventUniCode (965), GetKeyEventFlags (964), GetKeyEventCode (963), GetKeyEventChar (962), GetKeyEvent (962)

Listing: ./examples/kbdex/ex3.pp

```
Program Example3;

{ Program to demonstrate the GetKeyEventShiftState function. }

Uses keyboard;

Var
  K : TKeyEvent;
  S : Byte;

begin
  InitKeyBoard;
  Write('Press keys combined with CTRL/SHIFT/ALT');
  Writeln(', or press "q" to end.');
  Repeat
    K:=GetKeyEvent;
    K:=TranslateKeyEvent(K);
    S:=GetKeyEventShiftState(K);
    If (S=0) then
      Writeln('No special keys pressed')
    else
      begin
```

```
      WriteLn('Detected special keys : ',ShiftStateToString(K,False));
      WriteLn('Got key : ',KeyEventToString(K));
      end;
  Until (GetKeyEventChar(K)='q');
  DoneKeyboard;
end.
```

59.7.9 GetKeyEventUniCode

Synopsis: Return the Unicode key event.

Declaration: `function GetKeyEventUniCode(KeyEvent: TKeyEvent) : Word`

Visibility: default

Description: `GetKeyEventUniCode` returns the Unicode part of the given `KeyEvent` if it contains a translated Unicode character.

Errors: None.

See also: GetKeyEventShiftState (964), GetKeyEventFlags (964), GetKeyEventCode (963), GetKeyEventChar (962), GetKeyEvent (962)

59.7.10 InitKeyboard

Synopsis: Initialize the keyboard driver.

Declaration: `procedure InitKeyboard`

Visibility: default

Description: `InitKeyboard` initializes the keyboard driver. If the driver is already active, it does nothing. When the driver is initialized, it will do everything necessary to ensure the functioning of the keyboard, including allocating memory, initializing the terminal etc.

This function should be called once, before using any of the keyboard functions. When it is called, the DoneKeyboard (960) function should also be called before exiting the program or changing the keyboard driver with SetKeyboardDriver (969).

On Unix, the default keyboard driver sets terminal in raw mode. In raw mode the line feed behaves as an actual linefeed, i.e. the cursor is moved down one line. while the x coordinate does not change. To compensate, the default keyboard sets driver line ending of system.output to #13#10.

For an example, see most other functions.

Errors: None.

See also: DoneKeyboard (960), SetKeyboardDriver (969)

59.7.11 IsFunctionKey

Synopsis: Check whether a given event is a function key event.

Declaration: `function IsFunctionKey(KeyEvent: TKeyEvent) : Boolean`

Visibility: default

Description: `IsFunctionKey` returns `True` if the given key event in `KeyEvent` was a function key or not.

Errors: None.

See also: GetKeyEvent (962)

Listing: ./examples/kbdex/ex7.pp

```
program example1;

{ This program demonstrates the GetKeyEvent function }

uses keyboard;

Var
  K : TKeyEvent;

begin
  InitKeyBoard;
  Writeln('Press keys, press "q" to end.');
  Repeat
    K:=GetKeyEvent;
    K:=TranslateKeyEvent(K);
    If IsFunctionKey(K) then
      Writeln('Got function key : ',KeyEventToString(K))
    else
      Writeln('not a function key.');
  Until (GetKeyEventChar(K)='q');
  DoneKeyBoard;
end.
```

59.7.12 KeyEventToString

Synopsis: Return a string describing the key event.

Declaration: `function KeyEventToString(KeyEvent: TKeyEvent) : string`

Visibility: default

Description: `KeyEventToString` translates the key event in `KeyEvent` to a human-readable description of the pressed key. It will use the constants described in the constants section to do so.

For an example, see most other functions.

Errors: If an unknown key is passed, the scancode is returned, prefixed with the `SScanCode` string.

See also: FunctionKeyName (961), ShiftStateToString (969)

59.7.13 KeyPressed

Synopsis: Check event queue for key press.

Declaration: `function KeyPressed : Boolean`

Visibility: default

Description: `KeyPressed` checks the keyboard event queue to see whether a key event is present, and returns `True` if a key event is available. This function simply calls PollKeyEvent (967) and checks for a valid result.

Errors: None.

See also: PollKeyEvent (967), GetKeyEvent (962)

59.7.14 PollKeyEvent

Synopsis: Get next key event, but does not wait.

Declaration: `function PollKeyEvent : TKeyEvent`

Visibility: default

Description: `PollKeyEvent` checks whether a key event is available, and returns it if one is found. If no event is pending, it returns 0.

Note that this does not remove the key from the pending keys. The key should still be retrieved from the pending key events list with the GetKeyEvent (962) function.

Errors: None.

See also: PutKeyEvent (968), GetKeyEvent (962)

Listing: ./examples/kbdex/ex4.pp

```
program example4;

{ This program demonstrates the PollKeyEvent function }

uses keyboard;

Var
  K : TKeyEvent;

begin
  InitKeyBoard;
  Writeln('Press keys, press "q" to end.');
  Repeat
    K:=PollKeyEvent;
    If k<>0 then
      begin
      K:=GetKeyEvent;
      K:=TranslateKeyEvent(K);
      writeln;
      Writeln('Got key : ',KeyEventToString(K));
      end
    else
      write('.');
  Until (GetKeyEventChar(K)='q');
  DoneKeyBoard;
end.
```

59.7.15 PollShiftStateEvent

Synopsis: Check current shift state.

Declaration: `function PollShiftStateEvent : TKeyEvent`

Visibility: default

Description: `PollShiftStateEvent` returns the current shiftstate in a keyevent. This will return 0 if there is no key event pending.

Errors: None.

See also: PollKeyEvent (967), GetKeyEvent (962)

Listing: ./examples/kbdex/ex6.pp

```
program example6;

{ This program demonstrates the PollShiftStateEvent function }

uses keyboard;

Var
  K : TKeyEvent;

begin
  InitKeyBoard;
  Writeln('Press keys, press "q" to end.');
  Repeat
    K:=PollKeyEvent;
    If k<>0 then
      begin
      K:=PollShiftStateEvent;
      Writeln('Got shift state : ',ShiftStateToString(K,False));
      // Consume the key.
      K:=GetKeyEvent;
      K:=TranslateKeyEvent(K);
      end
{     else
      write('.')};
  Until (GetKeyEventChar(K)='q');
  DoneKeyBoard;
end.
```

59.7.16 PutKeyEvent

Synopsis: Put a key event in the event queue.

Declaration: `procedure PutKeyEvent(KeyEvent: TKeyEvent)`

Visibility: default

Description: `PutKeyEvent` adds the given `KeyEvent` to the input queue. Please note that depending on the implementation this can hold only one value, i.e. when calling `PutKeyEvent` multiple times, only the last pushed key will be remembered.

Errors: None

See also: PollKeyEvent (967), GetKeyEvent (962)

Listing: ./examples/kbdex/ex5.pp

```
program example5;

{ This program demonstrates the PutKeyEvent function }

uses keyboard;

Var
  K,k2 : TKeyEvent;
```

```
begin
  InitKeyBoard;
  Writeln('Press keys, press "q" to end.');
  K2:=0;
  Repeat
    K:=GetKeyEvent;
    If k<>0 then
      begin
      if (k2 mod 2)=0 then
        K2:=K+1
      else
        K2:=0;
      K:=TranslateKeyEvent(K);
      Writeln('Got key : ',KeyEventToString(K));
      if (K2<>0) then
        begin
        PutKeyEvent(k2);
        K2:=TranslateKeyEVent(K2);
        Writeln('Put key : ',KeyEventToString(K2))
        end
      end
  Until (GetKeyEventChar(K)='q');
  DoneKeyBoard;
end.
```

59.7.17 SetKeyboardDriver

Synopsis: Set a new keyboard driver.

Declaration: `function SetKeyboardDriver(const Driver: TKeyboardDriver) : Boolean`

Visibility: default

Description: `SetKeyBoardDriver` sets the keyboard driver to `Driver`, if the current keyboard driver is not yet initialized. If the current keyboard driver is initialized, then `SetKeyboardDriver` does nothing. Before setting the driver, the currently active driver should be disabled with a call to DoneKeyboard (960).

The function returns `True` if the driver was set, `False` if not.

For more information on setting the keyboard driver, see kbddriver (950).

Errors: None.

See also: GetKeyboardDriver (961), DoneKeyboard (960)

59.7.18 ShiftStateToString

Synopsis: Return description of key event shift state.

Declaration: `function ShiftStateToString(KeyEvent: TKeyEvent; UseLeftRight: Boolean) : string`

Visibility: default

Description: `ShiftStateToString` returns a string description of the shift state of the key event `KeyEvent`. This can be an empty string.

The shift state is described using the strings in the `SShift` constant.

For an example, see PollShiftStateEvent (967).

Errors: None.

See also: FunctionKeyName (961), KeyEventToString (966)

59.7.19 TranslateKeyEvent

Synopsis: Translate raw event to ascii key event.

Declaration: `function TranslateKeyEvent(KeyEvent: TKeyEvent) : TKeyEvent`

Visibility: default

Description: `TranslateKeyEvent` performs ASCII translation of the `KeyEvent`. It translates a physical key to a function key if the key is a function key, and translates the physical key to the ordinal of the ascii character if there is an equivalent character key.

For an example, see GetKeyEvent (962)

Errors: None.

See also: TranslateKeyEventUniCode (970)

59.7.20 TranslateKeyEventUniCode

Synopsis: Translate raw event to UNICode key event.

Declaration: `function TranslateKeyEventUniCode(KeyEvent: TKeyEvent) : TKeyEvent`

Visibility: default

Description: `TranslateKeyEventUniCode` performs Unicode translation of the `KeyEvent`. It is not yet implemented for all platforms.

Errors: If the function is not yet implemented, then the `ErrorCode` of the system unit will be set to `errKbdNotImplemented`

See also: TranslateKeyEvent (970)

59.8 TKeyboardDriver

```
TKeyboardDriver = record
  InitDriver : procedure;
  DoneDriver :
  procedure;
  GetKeyEvent : function : TKeyEvent;
  PollKeyEvent
  : function : TKeyEvent;
  GetShiftState : function : Byte;
  TranslateKeyEvent
  : function(KeyEvent: TKeyEvent) : TKeyEvent;
```

```
  TranslateKeyEventUniCode
  : function(KeyEvent: TKeyEvent) : TKeyEvent;
end
```

The `TKeyboardDriver` record can be used to install a custom keyboard driver with the SetKeyboardDriver (969) function.

The various fields correspond to the different functions of the keyboard unit interface. For more information about this record see kbddriver (950)

59.9 TKeyRecord

```
TKeyRecord = packed record
  Flags : Byte;
  ShiftState : Byte;
  KeyCode : Word;
end
```

The structure of a `TKeyRecord` structure is explained in the following table:

Table 59.4: Structure of TKeyRecord

Field	Meaning
KeyCode	Depending on `flags` either the physical representation of a key (under DOS scancode, ASCII code pair), or the
ShiftState	Shift-state when this key was pressed (or shortly after)
Flags	Determine how to interpret `KeyCode`

The shift-state can be checked using the various shift-state constants, and the flags in the last byte can be checked using one of the kbASCII, kbUniCode, kbFnKey, kbPhys, kbReleased constants.

If there are two keys returning the same char-code, there's no way to find out which one was pressed (Gray+ and Simple+). If it needs to be known which was pressed, the untranslated keycodes must be used, but these are system dependent. System dependent constants may be defined to cover those, with possibly having the same name (but different value).

59.10 TTreeElement

```
TTreeElement = record
  Next : PTreeElement;
  Parent : PTreeElement
  ;
  Child : PTreeElement;
  CanBeTerminal : Boolean;
  char : Byte
  ;
  ScanValue : Byte;
  CharValue : Byte;
  SpecialHandler : Tprocedure
  ;
end
```

`TTreeELement` is used to describe key scancode sequences, and is used to handle special key combinations in AddSpecialSequence (**??**) on UNIX platforms. There should be no need to handle records of this type.

Chapter 60

Reference for unit 'lineinfo'

60.1 Used units

Table 60.1: Used units by unit 'lineinfo'

Name	Page
System	1362

60.2 Overview

The lineinfo provides a routine that reads the debug information of an executable (if any exists) and returns source code information about this address. It works with `Stabs` debug information. Note that this unit is not thread-safe, and that its behaviour is undefined if multiple threads try to write a backtrace at the same time.

For DWARF debug information, the lnfodwrf (1006) unit must be used.

60.3 Constants, types and variables

60.3.1 Types

```
CodePointer = Pointer
```

`CodePointer` is added for 16-bit dos compatibility.

60.3.2 Variables

```
AllowReuseOfLineInfoData : Boolean = True
```

`AllowReuseOfLineInfoData` can be set to `True` to keep the last opened file open. When regularly creating backtraces (e.g. in a program log), this will significantly speed up operations.

60.4 Procedures and functions

60.4.1 CloseStabs

Synopsis: Close stabs info file descriptor.

Declaration: `procedure CloseStabs`

Visibility: default

Description: `CloseStabs` will close the file descriptor that was used to read STABS debug information. This is useful if AllowReuseOfLineInfoData (973) is used to cache STABS information.

Errors: None.

See also: AllowReuseOfLineInfoData (973)

60.4.2 GetLineInfo

Synopsis: Return source line information about an address.

Declaration:
```
function GetLineInfo(addr: PtrUInt; var func: string;
                     var source: string; var line: LongInt) : Boolean
```

Visibility: default

Description: `GetLineInfo` returns source line information about the address `addr`. It searches this information in the stabs debugging information found in the binary: If the file was compiled without debug information, nothing will be returned. Upon successful retrieval of the debug information, `True` is returned, and the `func` parameter is filled with the name of the function in which the address is located. The `source` parameter contains the name of the file in which the function was implemented, and `line` contains the line number in the source file for `addr`.

Errors: If no debug information is found, `False` is returned.

60.4.3 StabBackTraceStr

Synopsis: Get a backtrace from an address.

Declaration: `function StabBackTraceStr(addr: CodePointer) : string`

Visibility: default

Description: `StabBackTraceStr` returns a backtrace from a memory address `Addr`.

This is the actual callback for the backtrace handler System.BackTraceStrFunc (973).

Errors: None.

See also: GetLineInfo (974)

Chapter 61

Reference for unit 'Linux'

61.1 Used units

Table 61.1: Used units by unit 'Linux'

Name	Page
BaseUnix	146
System	1362
unixtype	2175

61.2 Overview

The linux unit contains Linux specific operating system calls.

The platform independent functionality of the FPC 1.0.X version of the linux unit has been split out over the UNIX (2135), baseunix (146) and unixutil (2191) units.

The X86-specific parts have been moved to the X86 (2272) unit.

61.3 Constants, types and variables

61.3.1 Constants

```
CAP_AUDIT_CONTROL = 30
```

Allow manipulation of kernel auditing features.

```
CAP_AUDIT_WRITE = 29
```

Allow writing to kernel audit log.

```
CAP_CHOWN = 0
```

Perform chown operation.

```
CAP_DAC_OVERRIDE = 1
```

Bypass file operation (rwx) checks.

```
CAP_DAC_READ_SEARCH = 2
```

Bypass file read-only operation checks.

```
CAP_FOWNER = 3
```

Bypass owner ID checks.

```
CAP_FSETID = 4
```

Do not clear SUID/GUID bits on modified files.

```
CAP_FS_MASK = $1f
```

?

```
CAP_IPC_LOCK = 14
```

Allow memory locking calls.

```
CAP_IPC_OWNER = 15
```

Bypass permission checks on IPC operations.

```
CAP_KILL = 5
```

Bypass permission checks for sending signals.

```
CAP_LEASE = 28
```

Allow file leases.

```
CAP_LINUX_IMMUTABLE = 9
```

Allow setting ext2 file attributes.

```
CAP_MKNOD = 27
```

Allow creation of special files through mknod calls.

```
CAP_NET_ADMIN = 12
```

Allow network operations (e.g. setting socket options).

```
CAP_NET_BIND_SERVICE = 10
```

Allow binding to ports less than 1024.

`CAP_NET_BROADCAST = 11`

Allow socket broadcast operations.

`CAP_NET_RAW = 13`

Allow use of RAW and PACKET sockets.

`CAP_SETGID = 6`

Allow GID manipulations.

`CAP_SETPCAP = 8`

Allow to set other process' capabilities.

`CAP_SETUID = 7`

Allow process ID manipulations.

`CAP_SYS_ADMIN = 21`

Allow various system administration calls.

`CAP_SYS_BOOT = 22`

Allow reboot calls.

`CAP_SYS_CHROOT = 18`

Allow chroot calls.

`CAP_SYS_MODULE = 16`

Allow loading/unloading of kernel modules.

`CAP_SYS_NICE = 23`

Allowing raising process and thread priorities.

`CAP_SYS_PACCT = 20`

Allow acct calls.

`CAP_SYS_PTRACE = 19`

Allow ptrace calls.

`CAP_SYS_RAWIO = 17`

Allow raw I/O port operations.

`CAP_SYS_RESOURCE = 24`

Allow use of special resources or raising of resource limits.

`CAP_SYS_TIME = 25`

Allow system or real-time clock modification.

`CAP_SYS_TTY_CONFIG = 26`

Allow vhangup calls.

`CLOCKS_MASK = CLOCK_REALTIME or CLOCK_MONOTONIC`

Mask for supported clocks.

`CLOCKS_MONO = CLOCK_MONOTONIC`

Monotonic clocks mask.

`CLOCK_MONOTONIC = 1`

Monotonic system time since some undetermined start point. Can change if time is set.

`CLOCK_MONOTONIC_COARSE = 6`

Less precise (but faster) version of `CLOCK_MONOTONIC`.

`CLOCK_MONOTONIC_RAW = 4`

Like `CLOCK_MONOTONIC`, not subject to NTP adjustments.

`CLOCK_PROCESS_CPUTIME_ID = 2`

Processs-specific high-resolution timer from the CPU.

`CLOCK_REALTIME = 0`

System wide real-time clock. Can only be set by root.

`CLOCK_REALTIME_COARSE = 5`

Less precise (but faster) version of `CLOCK_REALTIME`.

`CLOCK_SGI_CYCLE = 10`

High resolution timer.

`CLOCK_THREAD_CPUTIME_ID = 3`

Thread-specific high-resolution timer from the CPU.

```
CLONE_CHILD_CLEARTID = $00200000
```

Clone option: Erase child thread ID in child memory space when child exits.

```
CLONE_CHILD_SETTID = $01000000
```

Clone option: Store child thread ID in child memory.

```
CLONE_DETACHED = $00400000
```

Clone option: Start clone detached.

```
CLONE_FILES = $00000400
```

Clone (994) option: open files shared between processes.

```
CLONE_FS = $00000200
```

Clone (994) option: fs info shared between processes.

```
CLONE_NEWNS = $00020000
```

Clone options: Start child in new (file system) namespace.

```
CLONE_PARENT = $00008000
```

Clone options: Set child parent to parent of calling process.

```
CLONE_PARENT_SETTID = $00100000
```

Clone option: Store child thread ID in memory in both parent and child.

```
CLONE_PID = $00001000
```

Clone (994) option: PID shared between processes.

```
CLONE_PTRACE = $00002000
```

Clone options: if parent is traced, trace child also.

```
CLONE_SETTLS = $00080000
```

Clone option: The newtls parameter is the TLS descriptor of the child.

```
CLONE_SIGHAND = $00000800
```

Clone (994) option: signal handlers shared between processes.

```
CLONE_STOPPED = $02000000
```

Clone option: Start child in stopped state.

```
CLONE_SYSVSEM = $00040000
```

Clone option: Caller and child share the same semaphore undo values.

```
CLONE_THREAD = $00010000
```

Clone options: Set child in thread group of calling process.

```
CLONE_UNTRACED = $00800000
```

Clone option: Do not allow a ptrace call on this clone.

```
CLONE_VFORK = $00004000
```

Clone options: suspend parent till child execs.

```
CLONE_VM = $00000100
```

Clone (994) option: VM shared between processes.

```
CSIGNAL = $000000ff
```

Clone (994) option: Signal mask to be sent at exit.

```
EPOLLERR = $08
```

`event_wait` error condition on file descriptor.

```
EPOLLET = $80000000
```

Set `event_wait` edge trigger behaviour on file descriptor.

```
EPOLLHUP = $10
```

`event_wait` hang up event.

```
EPOLLIN = $01
```

`event_wait` input file descriptor ready event.

```
EPOLLONESHOT = $40000000
```

Set single-shot behaviour on `epoll_wait`.

```
EPOLLOUT = $04
```

`event_wait` output file descriptor ready event.

```
EPOLLPRI = $02
```

`event_wait` high priority data available on input file descriptor.

```
EPOLL_CTL_ADD = 1
```

Add filedescriptor to list of events.

```
EPOLL_CTL_DEL = 2
```

Delete event for filedescriptor.

```
EPOLL_CTL_MOD = 3
```

Modify event for filedescriptor.

```
FUTEX_CMP_REQUEUE = 4
```

Futex option: requeue waiting processes on other futex, but check it's value first.

```
FUTEX_FD = 2
```

Futex option: Associate file descriptor with futex.

```
FUTEX_LOCK_PI = 6
```

Futex option: Undocumented.

```
FUTEX_OP_ADD = 1
```

Futex operation: Undocumented.

```
FUTEX_OP_ANDN = 3
```

Futex operation: Undocumented.

```
FUTEX_OP_CMP_EQ = 0
```

Futex operation: Undocumented.

```
FUTEX_OP_CMP_GE = 5
```

Futex operation: Undocumented.

```
FUTEX_OP_CMP_GT = 4
```

Futex operation: Undocumented.

```
FUTEX_OP_CMP_LE = 3
```

Futex operation: Undocumented.

```
FUTEX_OP_CMP_LT = 2
```

Futex operation: Undocumented.

```
FUTEX_OP_CMP_NE = 1
```

Futex operation: Undocumented.

```
FUTEX_OP_OPARG_SHIFT = 8
```

Futex operation: Undocumented.

```
FUTEX_OP_OR = 2
```

Futex operation: Undocumented.

```
FUTEX_OP_SET = 0
```

Futex operation: Undocumented.

```
FUTEX_OP_XOR = 4
```

Futex operation: Undocumented.

```
FUTEX_REQUEUE = 3
```

Futex option: requeue waiting processes on other futex.

```
FUTEX_TRYLOCK_PI = 8
```

Futex option: Undocumented.

```
FUTEX_UNLOCK_PI = 7
```

Futex option: Undocumented.

```
FUTEX_WAIT = 0
```

Futex option: Wait on futex till wake call arrives.

```
FUTEX_WAKE = 1
```

Futex option: wakes any waiting processes on this futex.

```
FUTEX_WAKE_OP = 5
```

Futex option: Undocumented.

```
GIO_CMAP = $4B70
```

IOCTL: Get color palette on VGA+.

```
GIO_FONT = $4B60
```

IOCTL: Get font in expanded form.

```
GIO_FONTX = $4B6B
```

IOCTL: Get font in `consolefontdesc` record.

```
GIO_SCRNMAP = $4B40
```

IOCTL: get screen mapping from kernel.

```
GIO_UNIMAP = $4B66
```

IOCTL: get unicode-to-font mapping from kernel.

```
GIO_UNISCRNMAP = $4B69
```

IOCTL: get full Unicode screen mapping.

```
IN_ACCESS = $00000001
```

Data was read from file.

```
IN_ALL_EVENTS = IN_ACCESS or IN_MODIFY or IN_ATTRIB or IN_CLOSE or
  IN_OPEN or IN_MOVE or IN_CREATE or IN_DELETE or IN_DELETE_SELF or
  IN_MOVE_SELF
```

All possible events OR-ed together.

```
IN_ATTRIB = $00000004
```

File attributes changed.

```
IN_CLOEXEC = &02000000
```

`IN_CLOEXEC` can be set to indicate that the inotify file handle must be closed on exec.

```
IN_CLOSE = IN_CLOSE_WRITE or IN_CLOSE_NOWRITE
```

File was closed (read or write).

```
IN_CLOSE_NOWRITE = $00000010
```

File opened for read was closed.

```
IN_CLOSE_WRITE = $00000008
```

File opened for write was closed.

```
IN_CREATE = $00000100
```

A file was created in the directory.

```
IN_DELETE = $00000200
```

A file was deleted from the directory.

IN_DELETE_SELF = $00000400

Directory or file under observation was deleted.

IN_DONT_FOLLOW = $02000000

Do not follow symlinks.

IN_IGNORED = $00008000

Watch was ignored (removed). Only reported.

IN_ISDIR = $40000000

Event subject is a directory (reported only).

IN_MASK_ADD = $20000000

Add events to existing watch (OR-ing the sets) if one exists.

IN_MODIFY = $00000002

Data was written to file.

IN_MOVE = IN_MOVED_FROM or IN_MOVED_TO

File was moved (in or out of directory).

IN_MOVED_FROM = $00000040

File was moved away from watched directory.

IN_MOVED_TO = $00000080

File was moved into watched directory.

IN_MOVE_SELF = $00000800

Directory or file under observation was moved.

IN_NONBLOCK = &00004000

IN_NONBLOCK can be set to indicate that the inotify file handle should not block read operations.

IN_ONESHOT = $80000000

Only report one event, then remove the watch.

IN_ONLYDIR = $01000000

Only watch filename if it is a directory.

IN_OPEN = $00000020

File was opened.

IN_Q_OVERFLOW = $00004000

Queue overflowed. Only reported.

IN_UNMOUNT = $00002000

File system on which file resides was unmounted. Only reported.

KB_101 = 2

IOCTL: Keyboard types: 101 keys.

KB_84 = 1

IOCTL: Keyboard types: 84 keys.

KB_OTHER = 3

IOCTL: Keyboard types: other type.

KDADDIO = $4B34

IOCTL: add i/o port as valid.

KDDELIO = $4B35

IOCTL: delete i/o port as valid.

KDDISABIO = $4B37

IOCTL: disable i/o to video board.

KDENABIO = $4B36

IOCTL: enable i/o to video board.

KDFONTOP = $4B72

IOCTL: font operations.

KDGETKEYCODE = $4B4C

IOCTL: read kernel keycode table entry.

KDGETLED = $4B31

IOCTL: return current led state.

```
KDGETMODE = $4B3B
```

IOCTL: get current mode.

```
KDGKBDIACR = $4B4A
```

IOCTL: read kernel accent table.

```
KDGKBTYPE = $4B33
```

IOCTL: get keyboard type.

```
KDMAPDISP = $4B3C
```

IOCTL: map display into address space.

```
KDMKTONE = $4B30
```

IOCTL: generate tone.

```
KDSETKEYCODE = $4B4D
```

IOCTL: write kernel keycode table entry.

```
KDSETLED = $4B32
```

IOCTL: set led state.

```
KDSETMODE = $4B3A
```

IOCTL: set text/graphics mode.

```
KDSIGACCEPT = $4B4E
```

IOCTL: accept kbd generated signals.

```
KDSKBDIACR = $4B4B
```

IOCTL: write kernel accent table.

```
KDUNMAPDISP = $4B3D
```

IOCTL: unmap display from address space.

```
KD_GRAPHICS = 1
```

IOCTL: Tty modes: graphics mode.

```
KD_TEXT = 0
```

IOCTL: Tty modes: Text mode.

```
KD_TEXT0 = 2
```

IOCTL: Tty modes: Text mode (obsolete).

```
KD_TEXT1 = 3
```

IOCTL: Tty modes: Text mode (obsolete).

```
KIOCSOUND = $4B2F
```

IOCTL: start/stop sound generation (0 for off).

```
LED_CAP = 4
```

IOCTL: LED_CAP : caps lock led.

```
LED_NUM = 2
```

IOCTL: LED_SCR : Num lock led.

```
LED_SCR = 1
```

IOCTL: LED_SCR : scroll lock led.

```
LINUX_CAPABILITY_VERSION = $19980330
```

Current capability version in use by kernel.

```
MAP_DENYWRITE = $800
```

Read-only.

```
MAP_EXECUTABLE = $1000
```

Memory area is marked as executable.

```
MAP_GROWSDOWN = $100
```

Memory map grows down, like stack.

```
MAP_LOCKED = $2000
```

Memory pages are locked.

```
MAP_NORESERVE = $4000
```

Do not check for reservations.

```
MAX_CLOCKS = 16
```

Maximum number of clocks in the system.

```
MODIFY_LDT_CONTENTS_CODE = 2
```

Modify_ldt option: Undocumented.

```
MODIFY_LDT_CONTENTS_DATA = 0
```

Modify_ldt option: Undocumented.

```
MODIFY_LDT_CONTENTS_STACK = 1
```

Modify_ldt option: Undocumented.

```
O_CLOEXEC = $80000
```

Close on exec flag: close file handle on exec call.

```
PIO_CMAP = $4B71
```

IOCTL: Set color palette on VGA+.

```
PIO_FONT = $4B61
```

IOCTL: Use font in expanded form.

```
PIO_FONTRESET = $4B6D
```

IOCTL: Reset to default font.

```
PIO_FONTX = $4B6C
```

IOCTL: Set font in `consolefontdesc` record.

```
PIO_SCRNMAP = $4B41
```

IOCTL: put screen mapping table in kernel.

```
PIO_UNIMAP = $4B67
```

IOCTL: put unicode-to-font mapping in kernel.

```
PIO_UNIMAPCLR = $4B68
```

IOCTL: clear table, possibly advise hash algorithm.

```
PIO_UNISCRNMAP = $4B6A
```

IOCTL: set full Unicode screen mapping.

```
POLLMSG = $0400
```

Unused in Linux.

```
POLLRDHUP = $2000
```

Peer Shutdown/closed writing half of connection.

```
POLLREMOVE = $1000
```

Undocumented Linux extension of Poll.

```
SPLICE_F_GIFT = 8
```

Pages spliced in are a gift.

```
SPLICE_F_MORE = 4
```

Expect more data.

```
SPLICE_F_MOVE = 1
```

Move pages instead of copying.

```
SPLICE_F_NONBLOCK = 2
```

Don't block on pipe splicing operations.

```
STATX_ALL = $00000fff
```

```
STATX_ATIME = $00000020
```

```
STATX_ATTR_APPEND = $00000020
```

```
STATX_ATTR_AUTOMOUNT = $00001000
```

```
STATX_ATTR_COMPRESSED = $00000004
```

```
STATX_ATTR_ENCRYPTED = $00000800
```

```
STATX_ATTR_IMMUTABLE = $00000010
```

```
STATX_ATTR_NODUMP = $00000040
```

```
STATX_BASIC_STATS = $000007ff
```

```
STATX_BLOCKS = $00000400
```

```
STATX_BTIME = $00000800
```

```
STATX_CTIME = $00000080
```

```
STATX_GID = $00000010
```

```
STATX_INO = $00000100
```

```
STATX_MODE = $00000002
```

```
STATX_MTIME = $00000040
```

```
STATX_NLINK = $00000004
```

```
STATX_SIZE = $00000200
```

```
STATX_TYPE = $00000001
```

```
STATX_UID = $00000008
```

```
STATX__RESERVED = $80000000
```

```
SYNC_FILE_RANGE_WAIT_AFTER = 4
```

Wait upon write-out of specified pages in the range after performing any write.

```
SYNC_FILE_RANGE_WAIT_BEFORE = 1
```

Wait for write-out of previously-submitted specified pages before writing more data.

```
SYNC_FILE_RANGE_WRITE = 2
```

Initiate write of all dirty pages in the specified range.

```
UD_CONTENTS_CODE = MODIFY_LDT_CONTENTS_CODE shl 1
```

TLS segment descriptor: Undocumented.

```
UD_CONTENTS_DATA = MODIFY_LDT_CONTENTS_DATA shl 1
```

TLS segment descriptor: Undocumented.

```
UD_CONTENTS_STACK = MODIFY_LDT_CONTENTS_STACK shl 1
```

TLS segment descriptor: Undocumented.

```
UD_LIMIT_IN_PAGES = $10
```

TLS segment descriptor: Undocumented.

```
UD_LM = $80
```

TLS segment descriptor: Undocumented.

```
UD_READ_EXEC_ONLY = $08
```

TLS segment descriptor: Undocumented.

```
UD_SEG_32BIT = $01
```

TLS segment descriptor : Undocumented.

```
UD_SEG_NOT_PRESENT = $20
```

TLS segment descriptor: Undocumented.

```
UD_USEABLE = $40
```

TLS segment descriptor: Undocumented.

61.3.2 Types

```
clockid_t = cint
```

Clock id type.

```
EPoll_Data = record
case Integer of
0: (
  ptr : pointer;
);
1: (
  fd : cint;
);
2: (
  u32 : cuint;
);
3: (
  u64 : cuint64;
);
end
```

Data structure used in EPOLL IOCTL call.

```
kernel_time64_t = clonglong
```

```
PEPoll_Data = ^EPoll_Data
```

Pointer to EPoll_Data (992) record.

```
PEpoll_Event = ^EPoll_Event
```

Pointer to EPoll_Event (1003) type.

```
Pinotify_event = ^inotify_event
```

Pointer to inotify_event (1003) structure.

```
pkernel_timespec = ^kernel_timespec
```

```
pstatx = ^tstatx
```

```
pstatx_timestamp = ^statx_timestamp
```

```
PSysInfo = ^TSysInfo
```

Pointer to TSysInfo (1004) record.

```
Puser_cap_data = ^user_cap_data
```

Pointer to user_cap_data (1005) record.

```
Puser_cap_header = ^user_cap_header
```

Pointer to user_cap_header (1005) record.

```
PUser_Desc = ^user_desc
```

`PUser_Desc` is a pointer to the user_desc (1005) type.

```
TCloneFunc = function(args: pointer) : LongInt
```

Clone function prototype.

```
TEPoll_Data = EPoll_Data
```

Alias for EPoll_Data (992) type.

```
TEPoll_Event = EPoll_Event
```

Alias for EPoll_Event (1003) type.

```
tkernel_timespecs = Array[0..1] of kernel_timespec
```

```
TUser_Desc = user_desc
```

`TUser_Desc` is an alias for the user_desc (1005) type.

```
__s16 = SmallInt
```

```
__s32 = LongInt
```

```
__s64 = Int64
```

```
__u16 = Word
```

```
__u32 = DWord
```

```
__u64 = QWord
```

61.4 Procedures and functions

61.4.1 capget

Synopsis: Return the capabilities for the indicated thread.

Declaration: `function capget(header: Puser_cap_header; data: Puser_cap_data) : cint`

Visibility: default

Description: `capget` returns the capabilities of the indicated thread in `header`. The thread is identified by the process ID, or -1 for all caller (and child) process ID's.

Refer to the Linux man pages (7 capabilities) for more info.

Errors: On success, zero is returned, on error -1 is returned, and fperrno is set to the error.

See also: capset (993)

61.4.2 capset

Synopsis: Set the capabilities for the indicated thread.

Declaration: `function capset(header: Puser_cap_header; data: Puser_cap_data) : cint`

Visibility: default

Description: `capget` sets the capabilities of the indicated thread in `header`. The thread is identified by the process ID, or -1 for all caller (and child) process ID's.

Refer to the Linux man pages (7 capabilities) for more info.

Errors: On success, zero is returned, on error -1 is returned, and fperrno is set to the error.

See also: capget (993)

61.4.3 clock_getres

Synopsis: Get clock resolution.

Declaration:
```
function clock_getres(clk_id: clockid_t; res: ptimespec) : cint
```

Visibility: default

Description: `clock_getres` returns the resolution of the clock specified in `clk_id` in the `res` structure. It can be `Nil`. if the clock exists and the resolution can be retrieved, `0` is returned.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information.

See also: clock_gettime (994), clock_settime (994)

61.4.4 clock_gettime

Synopsis: Get the time of a clock.

Declaration:
```
function clock_gettime(clk_id: clockid_t; tp: ptimespec) : cint
```

Visibility: default

Description: `clock_gettime` returns the current time of the clock specified in `clk_id` in the `tp` structure. If the clock exists and the time can be retrieved, `0` is returned.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information.

See also: clock_getres (994), clock_settime (994)

61.4.5 clock_settime

Synopsis: Set the time of a clock.

Declaration:
```
function clock_settime(clk_id: clockid_t; tp: ptimespec) : cint
```

Visibility: default

Description: `clock_settime` sets the current time of the clock specified in `clk_id`. The time is specified in the `tp` structure. If the clock exists and the time can be retrieved, `0` is returned. The resolution is truncated to the resolution supported by the specified clock. Note that not all clocks can be set.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information.

See also: clock_getres (994), clock_gettime (994)

61.4.6 clone

Synopsis: Clone current process (create new thread).

Declaration:
```
function clone(func: TCloneFunc; sp: pointer; flags: LongInt;
                    args: pointer) : LongInt
```

Visibility: default

Description: `Clone` creates a child process which is a copy of the parent process, just like FpFork (195) does. In difference with `Fork`, however, the child process shares some parts of it's execution context with its parent, so it is suitable for the implementation of threads: many instances of a program that share the same memory.

When the child process is created, it starts executing the function `Func`, and passes it `Args`. The return value of `Func` is either the explicit return value of the function, or the exit code of the child process.

The `sp` pointer points to the memory reserved as stack space for the child process. This address should be the top of the memory block to be used as stack.

The `Flags` determine the behaviour of the `Clone` call. The low byte of the Flags contains the number of the signal that will be sent to the parent when the child dies. This may be bitwise OR'ed with the following constants:

CLONE_VM Parent and child share the same memory space, including memory (un)mapped with subsequent `mmap` calls.

CLONE_FS Parent and child have the same view of the file system; the `chroot`, `chdir` and `umask` calls affect both processes.

CLONE_FILESthe file descriptor table of parent and child is shared.

CLONE_SIGHANDthe parent and child share the same table of signal handlers. The signal masks are different, though.

CLONE_PID Parent and child have the same process ID.

Clone returns the process ID in the parent process, and -1 if an error occurred.

Errors: On error, -1 is returned to the parent, and no child is created.

sys_eagain Too many processes are running.

sys_enomem Not enough memory to create child process.

See also: #rtl.baseunix.FpFork (195)

61.4.7 epoll_create

Synopsis: Create new epoll file descriptor.

Declaration: `function epoll_create(size: cint) : cint`

Visibility: default

Description: `epoll_create` creates a new epoll file descriptor. The `size` argument indicates to the kernel approximately how many structures should be allocated, but is by no means an upper limit.

On success, a file descriptor is returned that can be used in subsequent epoll_ctl (996) or epoll_wait (996) calls, and should be closed using the fpClose (188) call.

Errors: On error, -1 is returned, and errno (198) is set.

See also: epoll_ctl (996), epoll_wait (996), fpClose (188)

61.4.8 epoll_ctl

Synopsis: Modify an epoll file descriptor.

Declaration:
```
function epoll_ctl(epfd: cint; op: cint; fd: cint; event: PEpoll_Event)
                        : cint
```

Visibility: default

Description: `epoll_ctl` performs the `op` operation on epoll file descriptor `epfd`. The operation will be monitored on file descriptor `fd`, and is optionally controlled by `event`.

`op` can be one of the following values:

EPOLL_CTL_ADD Add filedescriptor to list of events.

EPOLL_CTL_MOD Modify event for filedescriptor.

EPOLL_CTL_DEL Delete event for filedescriptor.

The `events` field in `event_data` is a bitmask of one or more of the following values:

EPOLLIN The file is ready for read operations

EPOLLOUT The file is ready for write operations.

EPOLLPRI Urgent data is available for read operations.

EPOLLERR An error condition is signaled on the file descriptor.

EPOLLHUP A Hang up happened on the file descriptor.

EPOLLET Set the Edge Triggered behaviour for the file descriptor.

EPOLLONESHOT Set One-Shot behaviour for the file descriptor. The event will be triggered only once.

Errors: On error -1 is returned, and errno is set accordingly.

See also: epoll_create (995), epoll_wait (996), fpClose (188)

61.4.9 epoll_wait

Synopsis: Wait for an event on an epoll file descriptor.

Declaration:
```
function epoll_wait(epfd: cint; events: PEpoll_Event; maxevents: cint;
                        timeout: cint) : cint
```

Visibility: default

Description: `epoll_wait` waits for `timeout` milliseconds for an event to occur on epoll file descriptor `epfd`. If `timeout` is -1, it waits indefinitely, if `timeour` is zero, it does not wait, but returns immediately, even if no events were detected.

On return, data for at most `maxevents` will be returned in the memory pointed to by `events`. The function returns the number of file descriptors for which events were reported. This can be zero if the timeout was reached.

Errors: On error -1 is returned, and errno is set accordingly.

See also: epoll_create (995), epoll_ctl (996), fpClose (188)

61.4.10 fdatasync

Synopsis: Synchronize the data in memory with the data on storage device.

Declaration:
```
function fdatasync(fd: cint) : cint
```

Visibility: default

Description: `fdatasync` does the same as `fpfsync` but does not flush the metadata, unless it is vital to the correct reading/writing of the file. In practice, this means that unless the file size changed, the file metadata will not be synced.

See also: #rtl.unix.fsync (2135)

61.4.11 futex

Synopsis: Perform a futex operation.

Declaration:
```
function futex(uaddr: pcint; op: cint; val: cint; timeout: ptimespec;
               addr2: pcint; val3: cint) : cint
function futex(var uaddr; op: cint; val: cint; timeout: ptimespec;
               var addr2; val3: cint) : cint
function futex(var uaddr; op: cint; val: cint; var timeout: TTimeSpec;
               var addr2; val3: cint) : cint
function futex(uaddr: pcint; op: cint; val: cint; timeout: ptimespec)
               : cint
function futex(var uaddr; op: cint; val: cint; timeout: ptimespec)
               : cint
function futex(var uaddr; op: cint; val: cint; var timeout: TTimeSpec)
               : cint
```

Visibility: default

Description: `futex` performs an operation on a memory futex as described in the kernel manual page for futex. The mutex is located at `uaddr`, the operation `op` is one of the following constants:

FUTEX_WAIT Futex option: Wait on futex till wake call arrives.

FUTEX_WAKE Futex option: Wait on futex till wake call arrives.

FUTEX_FD Futex option: Associate file descriptor with futex.

FUTEX_REQUEUE Futex option: requeue waiting processes on other futex.

FUTEX_CMP_REQUEUE Futex option: requeue waiting processes on other futex, but check it's value first.

The value to check for is indicated in `val`, and a timeout can be specified in `timeout`. The optional arguments `addr2` and `val3` are used only with the `FUTEX_REQUEUE` and `FUTEX_CMP_REQUEUE` operations.

In case of an error, -1 is return. All other return values must be interpreted according to the operation performed.

This call directly interfaces with the Linux kernel, more information can be found in the kernel manual pages.

Errors: On error, -1 is returned. Use #rtl.baseunix.fpgeterrno (198) to get the error code.

61.4.12 futex_op

Synopsis: Futex operation:.

Declaration: `function futex_op(op: cint; oparg: cint; cmp: cint; cmparg: cint) : cint`

Visibility: default

Description: `FUTEX_OP` Performs an operation on a futex:

```
FUTEX_OP := ((op and $F) shl 28) or
            ((cmp and $F) shl 24) or
            ((oparg and $FFF) shl 12)
            or (cmparg and $FFF);
```

61.4.13 futimens

Declaration: `function futimens(fd: cint; const times: tkernel_timespecs) : cint`

Visibility: default

61.4.14 inotify_add_watch

Synopsis: Add a watch to a notify file descriptor.

Declaration: `function inotify_add_watch(fd: cint; name: PChar; mask: cuint32) : cint`

Visibility: default

Description: `inotify_add_watch` can be used to add a watch to an initialized inotify file descriptor (`fd`). The file or directory to watch can be specified in the `name` parameter, and the events that must be reported can be specified in `mask`. The following flags can be specified:

IN_ACCESS Data was read from file.

IN_MODIFY Data was written to file.

IN_ATTRIB File attributes changed.

IN_CLOSE_WRITE File opened for write was closed

IN_CLOSE_NOWRITE File opened for read was closed

IN_CLOSE File was closed (read or write)

IN_OPEN File was opened

IN_MOVED_FROM File was moved away from watched directory

IN_MOVED_TO File was moved into watched directory

IN_MOVE File was moved (in or out of directory)

IN_CREATEA file was created in the directory.

IN_DELETEA file was deleted from the directory.

IN_DELETE_SELF Directory or file under observation was deleted.

IN_MOVE_SELF Directory or file under observation was moved.

IN_ALL_EVENTS All possible events OR-ed together.

These events can be OR-ed with some flags, controlling the behaviour of the watch:

IN_ONLYDIR Only watch filename if it is a directory.

IN_ISDIR Event occurred against directory.

IN_DONT_FOLLOW Do not follow symlinks

IN_MASK_ADD Add events to existing watch (OR-ing the sets) if one exists.

IN_ONESHOT Only report one event, then remove the watch.

On return, the function returns a watch descriptor, which will be reported in the inotify_event (1003) structure's `wd`.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information.

See also: inotify_init (999), inotify_init1 (999), inotify_rm_watch (999), inotify_event (1003)

61.4.15 inotify_init

Synopsis: Initialize a new inotify file descriptor.

Declaration: `function inotify_init : cint`

Visibility: default

Description: `inotify_init` initializes a new `INotify` file descriptor. No options can be specified. On return, the file descriptor is returned.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information

See also: inotify_init1 (999), inotify_add_watch (998), inotify_rm_watch (999)

61.4.16 inotify_init1

Synopsis: Initialize a new inotify file descriptor with extra options.

Declaration: `function inotify_init1(flags: cint) : cint`

Visibility: default

Description: `inotify_init1` initializes a new `INotify` file descriptor. The following options can be OR-ed and passed in flags:

IN_NONBLOCK Do not block on read.

IN_CLOEXEC Inotify close on exec flag.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information.

See also: inotify_init (999), inotify_add_watch (998), inotify_rm_watch (999)

61.4.17 inotify_rm_watch

Synopsis: Remove watch from Inotify file descriptor.

Declaration: `function inotify_rm_watch(fd: cint; wd: cint) : cint`

Visibility: default

Description: `inotify_rm_watch` removes watch descriptor `wd` from inotify descriptor `fd`. On success, 0 is returned.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information.

See also: inotify_init (999), inotify_init1 (999), inotify_add_watch (998), inotify_event (1003)

61.4.18 modify_ldt

Declaration: `function modify_ldt(func: cint; p: pointer; bytecount: culong) : cint`

Visibility: default

61.4.19 sched_yield

Synopsis: Yield the processor to another thread.

Declaration: `procedure sched_yield`

Visibility: default

Description: `sched_yield` yields the processor to another thread. The current thread is put at the back of its queue. If there is only 1 thread in the application, the thread continues to run. The call always returns zero.

61.4.20 setregid

Synopsis: Set Real and Effective Group ID.

Declaration: `function setregid(rgid: uid_t; egid: uid_t) : cint`

Visibility: default

Description: `setregid` sets the real group ID to `rgid` and the effective group ID to `egid`. Passing a value of -1 tells the system not to change that value.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information.

See also: setreuid (1000)

61.4.21 setreuid

Declaration: `function setreuid(ruid: uid_t; euid: uid_t) : cint`

Visibility: default

Description: `setreuid` sets the real user ID to `ruid` and the effective user ID to `euid`. Passing a value of -1 tells the system not to change that value.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information.

See also: setregid (1000)

61.4.22 statx

Declaration:
```
function statx(dfd: cint; filename: PChar; flags: cuint; mask: cuint;
                  var buf: tstatx) : cint
```

Visibility: default

61.4.23 sync_file_range

Synopsis: Force committing of data to disk.

Declaration:
```
function sync_file_range(fd: cint; offset: off64_t; nbytes: off64_t;
                         flags: cuint) : cint
```

Visibility: default

Description: `sync_file_range` forces the Linux kernel to write any data pages of a specified file (file descriptor `fd`) to disk. The range of the file is specified by the offset `offset` and the number of bytes `nbytes`. `Options` is an OR-ed combination of

SYNC_FILE_RANGE_WAIT_BEFORE Wait for write-out of previously-submitted specified pages before writing more data.

SYNC_FILE_RANGE_WRITE Initiate write of all dirty pages in the specified range.

SYNC_FILE_RANGE_WAIT_AFTER Wait upon write-out of specified pages in the range after performing any write.

If none is specified, the operation does nothing.

Errors: On return -1 is returned and fperrno is set to the actual error code. See the Linux man page for more on the error codes.

See also: fdatasync (997)

61.4.24 Sysinfo

Synopsis: Return kernel system information.

Declaration:
```
function Sysinfo(Info: PSysInfo) : cint
```

Visibility: default

Description: `SysInfo` returns system information in `Info`. Returned information in `Info` includes:

uptime Number of seconds since boot.

loads 1, 5 and 15 minute load averages.

totalram total amount of main memory.

freeram amount of free memory.

sharedram amount of shared memory.

bufferram amount of memory used by buffers.

totalswap total amount of swapspace.

freeswap amount of free swapspace.

procs number of current processes.

Errors: None.

See also: #rtl.baseunix.fpUname (234)

Listing: ./examples/linuxex/ex64.pp

```
program Example64;

{ Example to demonstrate the SysInfo function.
  Sysinfo is Linux-only. }

{$ifdef Linux}
Uses Linux;

Function Mb(L : Longint) : longint;

begin
  Mb:=L div (1024*1024);
end;

Var Info : TSysInfo;
    D,M,Secs,H : longint;
{$endif}

begin
  {$ifdef Linux}
  If Not (SysInfo(@Info)=0) then
    Halt(1);
  With Info do
    begin
    D:=Uptime div (3600*24);
    UpTime:=UpTime mod (3600*24);
    h:=uptime div 3600;
    uptime:=uptime mod 3600;
    m:=uptime div 60;
    secs:=uptime mod 60;
    Writeln('Uptime : ',d,'days, ',h,' hours, ',m,' min, ',secs,' s.');
    Writeln('Loads  : ',Loads[1],'/',Loads[2],'/',Loads[3]);
    Writeln('Total Ram  : ',Mb(totalram),'Mb.');
    Writeln('Free Ram   : ',Mb(freeram),'Mb.');
    Writeln('Shared Ram : ',Mb(sharedram),'Mb.');
    Writeln('Buffer Ram : ',Mb(bufferram),'Mb.');
    Writeln('Total Swap : ',Mb(totalswap),'Mb.');
    Writeln('Free Swap  : ',Mb(freeswap),'Mb.');
    end;
  {$endif}
end.
```

61.4.25 utimensat

Declaration: `function utimensat(dfd: cint; path: PChar; const times: tkernel_timespecs; flags: cint) : cint`

Visibility: default

61.5 EPoll_Event

```
EPoll_Event = packed record
  Events : cuint32;
  Data : TEPoll_Data
```

```
  ;
end
```

Structure used in epoll_ctl (996) call.

61.6 inotify_event

```
inotify_event = record
  wd : cint;
  mask : cuint32;
  cookie : cuint32
  ;
  len : cuint32;
  name : Char;
end
```

`inotify_event` is the structure used to report changes in a directory. When reading a inotify file descriptor, one or more `inotify_event` records can be read from the file descriptor.

61.7 kernel_timespec

```
kernel_timespec = record
  tv_sec : kernel_time64_t;
  tv_nsec : clonglong
  ;
end
```

61.8 statx_timestamp

```
statx_timestamp = record
  tv_sec : __s64;
  tv_nsec : __u32;
  __reserved
  : __s32;
end
```

61.9 tstatx

```
tstatx = record
  stx_mask : __u32;
  stx_blksize : __u32;
  stx_attributes
  : __u64;
  stx_nlink : __u32;
  stx_uid : __u32;
  stx_gid : __u32
```

```
  ;
  stx_mode : __u16;
  __spare0 : Array[0..0] of __u16;
  stx_ino
  : __u64;
  stx_size : __u64;
  stx_blocks : __u64;
  stx_attributes_mask
  : __u64;
  stx_atime : statx_timestamp;
  stx_btime : statx_timestamp
  ;
  stx_ctime : statx_timestamp;
  stx_mtime : statx_timestamp;
  stx_rdev_major : __u32;
  stx_rdev_minor : __u32;
  stx_dev_major
  : __u32;
  stx_dev_minor : __u32;
  __spare2 : Array[0..13] of __u64
  ;
end
```

61.10 TSysInfo

```
TSysInfo = record
  uptime : clong;
  loads : Array[0..2] of culong
  ;
  totalram : culong;
  freeram : culong;
  sharedram : culong;
  bufferram : culong;
  totalswap : culong;
  freeswap : culong;
  procs : cushort;
  pad : cushort;
  totalhigh : culong;
  freehigh
  : culong;
  mem_unit : cuint;
end
```

Record with system information, used by the SysInfo (1001) call.

61.11 user_cap_data

```
user_cap_data = record
  effective : cuint32;
  permitted : cuint32
```

```
  ;
  inheritable : cuint32;
end
```

`user_cap_data` describes the set of capabilities for the indicated thread.

61.12 user_cap_header

```
user_cap_header = record
  version : cuint32;
  pid : cint;
end
```

`user_cap_header` describes the root user capabilities for the current thread, as set by capget (993) and capset (993)

61.13 user_desc

```
user_desc = record
  entry_number : cuint;
  base_addr : cuint;
  limit : cuint;
  flags : cuint;
end
```

`user_desc` is the TLS (Thread Local Storage) segment descriptor used in the `Clone` call. It should not be used, as it contains highly kernel-specific data.

Chapter 62

Reference for unit 'lnfodwrf'

62.1 Used units

Table 62.1: Used units by unit 'lnfodwrf'

Name	Page
System	1362

62.2 Overview

The lnfodwrf provides a routine that reads the debug information of an executable (if any exists) and returns source code information about this address. It works with `DWARF` debug information. Note that this unit is not thread-safe, and that its behaviour is undefined if multiple threads try to write a backtrace at the same time.

For stabs debug information, the lineinfo (973) unit must be used.

62.3 Constants, types and variables

62.3.1 Types

```
CodePointer = Pointer
```

`CodePointer` is added for 16-bit dos compatibility.

62.3.2 Variables

```
AllowReuseOfLineInfoData : Boolean = True
```

`AllowReuseOfLineInfoData` can be set to `True` to keep the last opened file open. When regularly creating backtraces (e.g. in a program log), this will significantly speed up operations.

62.4 Procedures and functions

62.4.1 CloseDwarf

Synopsis: Close DWARF info file descriptor.

Declaration: `procedure CloseDwarf`

Visibility: default

Description: `CloseDwarf` will close the file descriptor that was used to read DWARF debug information. This is useful if AllowReuseOfLineInfoData (1006) is used to cache DWARF information.

Errors: None.

See also: AllowReuseOfLineInfoData (1006)

62.4.2 DwarfBackTraceStr

Synopsis: Get a backtrace from an address.

Declaration: `function DwarfBackTraceStr(addr: CodePointer) : string`

Visibility: default

Description: `DwarfBackTraceStr` returns a backtrace from a memory address `Addr`.

This is the actual callback for the backtrace handler System.BackTraceStrFunc (1006).

Errors: None.

See also: GetLineInfo (1007)

62.4.3 GetLineInfo

Synopsis: Return source line information about an address.

Declaration:
```
function GetLineInfo(addr: CodePtrUInt; var func: string;
                     var source: string; var line: LongInt) : Boolean
```

Visibility: default

Description: `GetLineInfo` returns source line information about the address `addr`. It searches this information in the DWARF debugging information found in the binary: If the file was compiled without debug information, nothing will be returned. Upon successful retrieval of the debug information, `True` is returned, and the `func` parameter is filled with the name of the function in which the address is located. The `source` parameter contains the name of the file in which the function was implemented, and `line` contains the line number in the source file for `addr`.

Errors: If no debug information is found, `False` is returned.

Chapter 63

Reference for unit 'Math'

63.1 Used units

Table 63.1: Used units by unit 'Math'

Name	Page
System	1362
sysutils	1635

63.2 Overview

This document describes the math unit. The `math` unit was initially written by Florian Klaempfl. It provides mathematical functions which aren't covered by the system unit.

This chapter starts out with a definition of all types and constants that are defined, after which an overview is presented of the available functions, grouped by category, and the last part contains a complete explanation of each function.

The following things must be taken into account when using this unit:

1. This unit is compiled in Object Pascal mode so all `integers` are 32 bit.
2. Some overloaded functions exist for data arrays of integers and floats. When using the address operator (`@`) to pass an array of data to such a function, make sure the address is typecasted to the right type, or turn on the 'typed address operator' feature. failing to do so, will cause the compiler not be able to decide which function you want to call.

63.3 Cash flow functions.

The cash flow functions in the math unit resolve the following equation:

```
FV + PV * q^n + PMT (q^n - 1) / (q - 1) = 0
```

In this formula, the following variables are present:

FV Future value

PV Present value

PMT Payment per period

n Number of payments (number of periods)

q> Interest Rate (return rate)

The financial functions FutureValue (1025), NumberOfPeriods (1041), Payment (1042), PresentValue (1045) and InterestRate (1028) solve this equation for one of the variables, when the other variables are known.

See also: FutureValue (1025), NumberOfPeriods (1041), Payment (1042), PresentValue (1045), TPaymentTime (1013)

63.4 Geometrical functions.

Table 63.2:

Name	Description
hypot (1027)	Hypotenuse of triangle
norm (1041)	Euclidean norm

63.5 Statistical functions.

Table 63.3:

Name	Description
mean (1035)	Mean of values
meanandstddev (1036)	Mean and standard deviation of values
momentskewkurtosis (1040)	Moments, skew and kurtosis
popnstddev (1042)	Population standard deviation
popnvariance (1043)	Population variance
randg (1047)	Gaussian distributed random value
stddev (1052)	Standard deviation
sum (1053)	Sum of values
sumofsquares (1054)	Sum of squared values
sumsandsquares (1055)	Sum of values and squared values
totalvariance (1057)	Total variance of values
variance (1058)	variance of values

63.6 Number converting.

Table 63.4:

Name	Description
ceil (1018)	Round to infinity
floor (1023)	Round to minus infinity
frexp (1025)	Return mantissa and exponent

63.7 Exponential and logarithmic functions.

Table 63.5:

Name	Description
intpower (1029)	Raise float to integer power
ldexp (1030)	Calculate $2ĝ x$
lnxp1 (1031)	calculate `log(x+1)`
log10 (1031)	calculate 10-base log
log2 (1032)	calculate 2-base log
logn (1032)	calculate N-base log
power (1044)	raise float to arbitrary power

63.8 Hyperbolic functions.

Table 63.6:

Name	Description
arcosh (1015)	calculate reverse hyperbolic cosine
arsinh (1017)	calculate reverse hyperbolic sine
artanh (1017)	calculate reverse hyperbolic tangent
cosh (1020)	calculate hyperbolic cosine
sinh (1052)	calculate hyperbolic sine
tanh (1056)	calculate hyperbolic tangent

63.9 Trigonometric functions.

Table 63.7:

Name	Description
arccos (1014)	calculate reverse cosine
arcsin (1015)	calculate reverse sine
arctan2 (1016)	calculate reverse tangent
cotan (1020)	calculate cotangent
sincos (1051)	calculate sine and cosine
tan (1056)	calculate tangent

63.10 Angle unit conversion.

Routines to convert angles between different angle units.

Table 63.8:

Name	Description
cycletorad (1021)	convert cycles to radians
degtograd (1022)	convert degrees to grads
degtorad (1022)	convert degrees to radians
gradtodeg (1026)	convert grads to degrees
gradtorad (1027)	convert grads to radians
radtocycle (1045)	convert radians to cycles
radtodeg (1046)	convert radians to degrees
radtograd (1046)	convert radians to grads

63.11 Min/max determination.

Functions to determine the minimum or maximum of numbers:

Table 63.9:

Name	Description
max (1033)	Maximum of 2 values
maxIntValue (1033)	Maximum of an array of integer values
maxvalue (1034)	Maximum of an array of values
min (1037)	Minimum of 2 values
minIntValue (1038)	Minimum of an array of integer values
minvalue (1038)	Minimum of an array of values

63.12 Constants, types and variables

63.12.1 Constants

```
EqualsValue = 0
```

Values are the same.

```
GreaterThanValue = High(TValueRelationship)
```

First values is greater than second value.

```
Infinity = 1.0 / 0.0
```

Value is infinity.

```
LessThanValue = Low(TValueRelationship)
```

First value is less than second value.

```
MaxFloat = 0
```

Maximum value of float type.

```
MinFloat = 0
```

Minimum value (closest to zero) of float type.

```
NaN = 0.0 / 0.0
```

Value is Not a Number.

```
NegativeValue = Low(TValueSign)
```

Value is negative.

```
NegInfinity = (- 1.0) / (0.0)
```

Value is negative (minus) infinity.

```
PositiveValue = High(TValueSign)
```

Value is positive.

```
ZeroValue = 0
```

Value is zero.

63.12.2 Types

```
Float = MaxFloatType
```

All calculations are done with the Float type which is the largest float type available for the current CPU. This allows to recompile the unit with a different float type to obtain a desired precision. The pointer type PFloat (1013) is used in functions that accept an array of values of arbitrary length.

```
PFloat = ^Float
```

Pointer to Float (1013) type.

```
PInteger = ObjPas.PInteger
```

Pointer to integer type.

```
TFPUException = system.TFPUException
```

Type describing Floating Point processor exceptions.

```
TFPUExceptionMask = system.TFPUExceptionMask
```

Type to set the Floating Point Unit exception mask.

```
TFPUPrecisionMode = system.TFPUPrecisionMode
```

Type describing the default precision for the Floating Point processor.

```
TFPURoundingMode = system.TFPURoundingMode
```

Type describing the rounding mode for the Floating Point processor.

```
TPaymentTime = (ptEndOfPeriod,ptStartOfPeriod)
```

Table 63.10: Enumeration values for type TPaymentTime

Value	Explanation
ptEndOfPeriod	End of period.
ptStartOfPeriod	Start of period.

Type used in financial (interest) calculations.

```
TRoundToRange = - 37..37
```

`TRoundToRange` is the range of valid digits to be used in the RoundTo (1048) function.

```
TValueRelationship = - 1..1
```

Type to describe relational order between values.

```
TValueSign = - 1..1
```

Type indicating sign of a value.

63.13 Procedures and functions

63.13.1 ArcCos

Synopsis: Return inverse cosine.

Declaration: `function ArcCos(x: Float) : Float`

Visibility: default

Description: `Arccos` returns the inverse cosine of its argument `x`. The argument `x` should lie between -1 and 1 (borders included).

Errors: If the argument `x` is not in the allowed range, an `EInvalidArgument` exception is raised.

See also: arcsin (1015), arcosh (1015), arsinh (1017), artanh (1017)

Listing: ./examples/mathex/ex1.pp

```
Program Example1;

{ Program to demonstrate the arccos function. }

Uses math;

  Procedure WriteRadDeg(X : float);

  begin
    WriteIn(X:8:5,' rad = ',radtodeg(x):8:5,' degrees.')
  end;

begin
  WriteRadDeg (arccos(1));
  WriteRadDeg (arccos(sqrt(3)/2));
  WriteRadDeg (arccos(sqrt(2)/2));
  WriteRadDeg (arccos(1/2));
  WriteRadDeg (arccos(0));
  WriteRadDeg (arccos(-1));
end.
```

63.13.2 ArcCosH

Synopsis: Return inverse hyperbolic cosine.

Declaration: `function ArcCosH(x: Float) : Float`

Visibility: default

Description: `arccosh` returns the inverse hyperbolic cosine of it's argument `x`.

This function is an alias for arcosh (1015), provided for Delphi compatibility.

See also: arcosh (1015)

63.13.3 ArCosH

Synopsis: Return inverse hyperbolic cosine.

Declaration: `function ArCosH(x: Float) : Float`

Visibility: default

Description: `Arcosh` returns the inverse hyperbolic cosine of its argument `x`. The argument `x` should be larger than 1. The `arccosh` variant of this function is supplied for Delphi compatibility.

Errors: If the argument `x` is not in the allowed range, an `EInvalidArgument` exception is raised.

See also: cosh (1020), sinh (1052), arcsin (1015), arsinh (1017), artanh (1017), tanh (1056)

Listing: ./examples/mathex/ex3.pp

```
Program Example3;

{ Program to demonstrate the arcosh function. }

Uses math;

begin
  Writeln(arcosh(1));
  Writeln(arcosh(2));
end.
```

63.13.4 ArcSin

Synopsis: Return inverse sine.

Declaration: `function ArcSin(x: Float) : Float`

Visibility: default

Description: `Arcsin` returns the inverse sine of its argument `x`. The argument `x` should lie between -1 and 1.

Errors: If the argument `x` is not in the allowed range, an `EInvalidArgument` exception is raised.

See also: arccos (1014), arcosh (1015), arsinh (1017), artanh (1017)

Listing: ./examples/mathex/ex2.pp

```
Program Example1;

{ Program to demonstrate the arcsin function. }

Uses math;

  Procedure WriteRadDeg(X : float);

  begin
    Writeln(X:8:5,' rad = ',radtodeg(x):8:5,' degrees.')
  end;

begin
  WriteRadDeg (arcsin(1));
  WriteRadDeg (arcsin(sqrt(3)/2));
```

```
  WriteRadDeg (arcsin(sqrt(2)/2));
  WriteRadDeg (arcsin(1/2));
  WriteRadDeg (arcsin(0));
  WriteRadDeg (arcsin(-1));
end.
```

63.13.5 ArcSinH

Synopsis: Return inverse hyperbolic sine.

Declaration: `function ArcSinH(x: Float) : Float`

Visibility: default

Description: `arcsinh` returns the inverse hyperbolic sine of it's argument `x`.

This function is an alias for arsinh (1017), provided for Delphi compatibility.

See also: arsinh (1017)

63.13.6 ArcTan2

Synopsis: Return arctangent of (y/x).

Declaration: `function ArcTan2(y: Float; x: Float) : Float`

Visibility: default

Description: `arctan2` calculates `arctan(y/x)`, and returns an angle in the correct quadrant. The returned angle will be in the range $-\pi$ to π radians. The values of `x` and `y` must be between -2\^{}64 and 2\^{}64, moreover `x` should be different from zero. On Intel systems this function is implemented with the native intel `fpatan` instruction.

See also: arccos (1014), arcosh (1015), arsinh (1017), artanh (1017)

Listing: ./examples/mathex/ex6.pp

```
Program Example6;

{ Program to demonstrate the arctan2 function. }

Uses math;

  Procedure WriteRadDeg(X : float);

  begin
    WriteIn(X:8:5,' rad = ',radtodeg(x):8:5,' degrees.')
  end;

begin
  WriteRadDeg (arctan2(2,1));
end.
```

63.13.7 ArcTanH

Synopsis: Return inverse hyperbolic tangent.

Declaration: `function ArcTanH(x: Float) : Float`

Visibility: default

Description: `arcsinh` returns the inverse hyperbolic tangent of it's argument `x`.

This function is an alias for artanh (1017), provided for Delphi compatibility.

See also: artanh (1017)

63.13.8 ArSinH

Synopsis: Return inverse hyperbolic sine.

Declaration: `function ArSinH(x: Float) : Float`

Visibility: default

Description: `arsinh` returns the inverse hyperbolic sine of its argument `x`. The `arscsinh` variant of this function is supplied for Delphi compatibility.

Errors: None.

See also: arcosh (1015), arccos (1014), arcsin (1015), artanh (1017)

Listing: ./examples/mathex/ex4.pp

```
Program Example4;

{ Program to demonstrate the arsinh function. }

Uses math;

begin
  Writeln(arsinh(0));
  Writeln(arsinh(1));
end.
```

63.13.9 ArTanH

Synopsis: Return inverse hyperbolic tangent.

Declaration: `function ArTanH(x: Float) : Float`

Visibility: default

Description: `artanh` returns the inverse hyperbolic tangent of its argument `x`, where `x` should lie in the interval [-1,1], borders included. The `arctanh` variant of this function is supplied for Delphi compatibility.

Errors: In case `x` is not in the interval [-1,1], an `EInvalidArgument` exception is raised.

See also: arcosh (1015), arccos (1014), arcsin (1015), artanh (1017)

Listing: ./examples/mathex/ex5.pp

```
Program Example5;

{ Program to demonstrate the artanh function. }

Uses math;

begin
  WriteLn(artanh(0));
  WriteLn(artanh(0.5));
end.
```

63.13.10 Ceil

Synopsis: Return the lowest integer number greater than or equal to argument.

Declaration: `function Ceil(x: Float) : Integer`

Visibility: default

Description: `Ceil` returns the lowest integer number greater than or equal to `x`. The absolute value of `x` should be less than `maxint`.

Errors: If the absolute value of `x` is larger than maxint, an overflow error will occur.

See also: floor (1023)

Listing: ./examples/mathex/ex7.pp

```
Program Example7;

{ Program to demonstrate the Ceil function. }

Uses math;

begin
  WriteLn(Ceil(-3.7)); // should be -3
  WriteLn(Ceil(3.7));  // should be 4
  WriteLn(Ceil(-4.0)); // should be -4
end.
```

63.13.11 Ceil64

Synopsis: Round to the nearest bigger int64 value.

Declaration: `function Ceil64(x: Float) : Int64`

Visibility: default

Description: `Ceil64` rounds the value `X`to the nearest bigger int64 value. The result is always bigger than `X`.

Errors: None.

See also: ceil (1018), floor64 (1024)

63.13.12 ClearExceptions

Synopsis: Clear Floating Point Unit exceptions.

Declaration:
```
procedure ClearExceptions(RaisePending: Boolean)
```

Visibility: default

Description: Clear Floating Point Unit exceptions.

63.13.13 CompareValue

Synopsis: Compare 2 values.

Declaration:
```
function CompareValue(const A: Integer; const B: Integer)
                        : TValueRelationship
function CompareValue(const A: Int64; const B: Int64)
                        : TValueRelationship
function CompareValue(const A: QWord; const B: QWord)
                        : TValueRelationship
function CompareValue(const A: Single; const B: Single; delta: Single)
                        : TValueRelationship
function CompareValue(const A: Double; const B: Double; delta: Double)
                        : TValueRelationship
function CompareValue(const A: Extended; const B: Extended;
                      delta: Extended) : TValueRelationship
```

Visibility: default

Description: `CompareValue` compares 2 integer or floating point values `A` and `B` and returns one of the following values:

-1if `A<B`

0if `A=B`

1if `A>B`

See also: TValueRelationship (1013)

63.13.14 Cosecant

Synopsis: Calculate cosecant.

Declaration:
```
function Cosecant(x: Float) : Float
```

Visibility: default

Description: `cosecant` calculates the cosecant (1/`sin(x)`) of its argument `x`.

Errors: If 0 or 180 degrees is specified, an exception will be raised.

See also: secant (1049)

63.13.15 CosH

Synopsis: Return hyperbolic cosine.

Declaration: `function CosH(x: Float) : Float`

Visibility: default

Description: `Cosh` returns the hyperbolic cosine of it's argument {x}.

Errors: None.

See also: arcosh (1015), sinh (1052), arsinh (1017)

Listing: ./examples/mathex/ex8.pp

```
Program Example8;

{ Program to demonstrate the cosh function. }

Uses math;

begin
  WriteLn(Cosh(0));
  WriteLn(Cosh(1));
end.
```

63.13.16 Cot

Synopsis: Alias for `Cotan`.

Declaration: `function Cot(x: Float) : Float`

Visibility: default

Description: `cot` is an alias for the cotan (1020) function.

See also: cotan (1020)

63.13.17 Cotan

Synopsis: Return cotangent.

Declaration: `function Cotan(x: Float) : Float`

Visibility: default

Description: `Cotan` returns the cotangent of it's argument `x`. The argument `x` must be in radians. `x` should be different from zero.

Errors: If `x` is zero then a overflow error will occur.

See also: tanh (1056)

Listing: ./examples/mathex/ex9.pp

```
Program Example9;

{ Program to demonstrate the cotan function. }

Uses math;

begin
  writeln(cotan(pi/2));
  Writeln(cotan(pi/3));
  Writeln(cotan(pi/4));
end.
```

63.13.18 Csc

Synopsis: Alias for `cosecant`.

Declaration: `function Csc(x: Float) : Float`

Visibility: default

Description: `csc` is an alias for the cosecant (1019) function.

See also: cosecant (1019)

63.13.19 CycleToRad

Synopsis: Convert cycle angle to radians angle.

Declaration: `function CycleToRad(cycle: Float) : Float`

Visibility: default

Description: `Cycletorad` transforms it's argument `cycle` (an angle expressed in cycles) to radians. (1 cycle is 2π radians).

Errors: None.

See also: degtograd (1022), degtorad (1022), radtodeg (1046), radtograd (1046), radtocycle (1045)

Listing: ./examples/mathex/ex10.pp

```
Program Example10;

{ Program to demonstrate the cycletorad function. }

Uses math;

begin
  writeln(cos(cycletorad(1/6))); // Should print 1/2
  writeln(cos(cycletorad(1/8))); // should be sqrt(2)/2
end.
```

63.13.20 DegNormalize

Synopsis: Normalize an angle measured in degrees.

Declaration:
```
function DegNormalize(deg: single) : single
function DegNormalize(deg: Double) : Double
function DegNormalize(deg: extended) : extended
```

Visibility: default

Description: `DegNormalize` returns the angle `deg` as a positive angle between `0` and `360` degrees.

63.13.21 DegToGrad

Synopsis: Convert degree angle to grads angle.

Declaration:
```
function DegToGrad(deg: Float) : Float
```

Visibility: default

Description: `Degtograd` transforms it's argument `deg` (an angle in degrees) to grads. (90 degrees is 100 grad.)

Errors: None.

See also: cycletorad (1021), degtorad (1022), radtodeg (1046), radtograd (1046), radtocycle (1045)

Listing: ./examples/mathex/ex11.pp

```
Program Example11;

{ Program to demonstrate the degtograd function. }

Uses math;

begin
  writeln(degtograd(90));
  writeln(degtograd(180));
  writeln(degtograd(270))
end.
```

63.13.22 DegToRad

Synopsis: Convert degree angle to radians angle.

Declaration:
```
function DegToRad(deg: Float) : Float
```

Visibility: default

Description: `Degtorad` converts it's argument `deg` (an angle in degrees) to radians. (pi radians is 180 degrees)

Errors: None.

See also: cycletorad (1021), degtograd (1022), radtodeg (1046), radtograd (1046), radtocycle (1045)

Listing: ./examples/mathex/ex12.pp

```
Program Example12;

{ Program to demonstrate the degtorad function. }

Uses math;

begin
  writeln(degtorad(45));
  writeln(degtorad(90));
  writeln(degtorad(180));
  writeln(degtorad(270));
  writeln(degtorad(360));
end.
```

63.13.23 DivMod

Synopsis: Return DIV and MOD of arguments.

Declaration:
```
procedure DivMod(Dividend: LongInt; Divisor: Word; var Result: Word;
                 var Remainder: Word)
procedure DivMod(Dividend: LongInt; Divisor: Word;
                 var Result: SmallInt; var Remainder: SmallInt)
procedure DivMod(Dividend: DWord; Divisor: DWord; var Result: DWord;
                 var Remainder: DWord)
procedure DivMod(Dividend: LongInt; Divisor: LongInt;
                 var Result: LongInt; var Remainder: LongInt)
```

Visibility: default

Description: `DivMod` returns `Dividend` DIV `Divisor` in `Result`, and `Dividend` MOD `Divisor` in `Remainder`

63.13.24 EnsureRange

Synopsis: Change value so it fits in a specified range.

Declaration:
```
function EnsureRange(const AValue: Integer; const AMin: Integer;
                     const AMax: Integer) : Integer;  Overload
function EnsureRange(const AValue: Int64; const AMin: Int64;
                     const AMax: Int64) : Int64;  Overload
function EnsureRange(const AValue: Double; const AMin: Double;
                     const AMax: Double) : Double;  Overload
```

Visibility: default

Description: `EnsureRange` returns `Value` if `AValue` is in the range `AMin`..`AMax`. It returns `AMin` if the value is less than `AMin`, or `AMax` if the value is larger than `AMax`.

See also: InRange (1028)

63.13.25 Floor

Synopsis: Return the largest integer smaller than or equal to argument.

Declaration:
```
function Floor(x: Float) : Integer
```

Visibility: default

Description: `Floor` returns the largest integer smaller than or equal to `x`. The absolute value of `x` should be less than `maxint`.

Errors: If `x` is larger than `maxint`, an overflow will occur.

See also: ceil (1018)

Listing: ./examples/mathex/ex13.pp

```
Program Example13;

{ Program to demonstrate the floor function. }

Uses math;

begin
  Writeln(Floor(-3.7)); // should be -4
  Writeln(Floor(3.7));  // should be 3
  Writeln(Floor(-4.0)); // should be -4
end.
```

63.13.26 Floor64

Synopsis: Round to the nearest smaller int64 value.

Declaration: `function Floor64(x: Float) : Int64`

Visibility: default

Description: `Floor64` rounds the value `X`to the nearest smaller int64 value. The result is always smaller than `X`.

Errors: None.

See also: floor (1023), ceil64 (1018)

63.13.27 FMod

Synopsis: Floatin point modulo.

Declaration:
```
function FMod(const a: Single; const b: Single) : Single;  Overload
function FMod(const a: Double; const b: Double) : Double;  Overload
function FMod(const a: Extended; const b: Extended) : Extended
             ;  Overload
```

Visibility: default

Description: `FMod` is the floating-point equivalent of the modulo operation `a mod b`. It returns the result of `a-b * Int(a/b)`

Errors: `b` may not be zero, but no check is performed.

63.13.28 Frexp

Synopsis: Return mantissa and exponent.

Declaration: `procedure Frexp(X: Float; var Mantissa: Float; var Exponent: Integer)`

Visibility: default

Description: `Frexp` returns the mantissa and exponent of it's argument `x` in `mantissa` and `exponent`.

Errors: None

Listing: ./examples/mathex/ex14.pp

```
Program Example14;

{ Program to demonstrate the frexp function. }

Uses math;

Procedure dofrexp(Const X : extended);

var man : extended;
    exp: longint;

begin
  man:=0;
  exp:=0;
  frexp(x,man,exp);
  write(x,' has ');
  Writeln('mantissa ',man,' and exponent ',exp);
end;


begin
//   dofrexp(1.00);
   dofrexp(1.02e-1);
   dofrexp(1.03e-2);
   dofrexp(1.02e1);
   dofrexp(1.03e2);
end.
```

63.13.29 FutureValue

Synopsis: Calculate the future value of an investment.

Declaration: `function FutureValue(ARate: Float; NPeriods: Integer; APayment: Float;
                          APresentValue: Float; APaymentTime: TPaymentTime)
                           : Float`

Visibility: default

Description: `FutureValue` calculates the future value of an investment (`FV`) in the cash flow formula (see CashFlowFunctions (1008)) The function result is the future value of an investment of `APresentValue` (`PV`), where `APayment` (`PMT`) is invested for `NPeriods` (`n`) at the rate of `ARate` (`q`) per period.

The `APaymentTime` parameter determines whether the investment (payment) is an ordinary annuity or an annuity due: `ptEndOfPeriod` must be used if payments are at the end of each period. If the payments are at the beginning of the periode, `ptStartOfPeriod` must be used.

See also: InterestRate (1028), NumberOfPeriods (1041), Payment (1042), PresentValue (1045), TPaymentTime (1013), CashFlowFunctions (1008)

63.13.30 GetExceptionMask

Synopsis: Get the Floating Point Unit exception mask.

Declaration: `function GetExceptionMask : TFPUExceptionMask`

Visibility: default

Description: Get the Floating Point Unit exception mask.

63.13.31 GetPrecisionMode

Synopsis: Return the Floating Point Unit precision mode.

Declaration: `function GetPrecisionMode : TFPUPrecisionMode`

Visibility: default

Description: Return the Floating Point Unit precision mode.

63.13.32 GetRoundMode

Synopsis: Return the Floating Point Unit rounding mode.

Declaration: `function GetRoundMode : TFPURoundingMode`

Visibility: default

Description: Return the Floating Point Unit rounding mode.

63.13.33 GradToDeg

Synopsis: Convert grads angle to degrees angle.

Declaration: `function GradToDeg(grad: Float) : Float`

Visibility: default

Description: `Gradtodeg` converts its argument `grad` (an angle in grads) to degrees. (100 grad is 90 degrees)

Errors: None.

See also: cycletorad (1021), degtograd (1022), radtodeg (1046), radtograd (1046), radtocycle (1045), gradtorad (1027)

Listing: ./examples/mathex/ex15.pp

```
Program Example15;

{ Program to demonstrate the gradtodeg function. }

Uses math;

begin
```

```
  writeln(gradtodeg(100));
  writeln(gradtodeg(200));
  writeln(gradtodeg(300));
end.
```

63.13.34 GradToRad

Synopsis: Convert grads angle to radians angle.

Declaration: `function GradToRad(grad: Float) : Float`

Visibility: default

Description: `Gradtorad` converts its argument `grad` (an angle in grads) to radians. (200 grad is pi degrees).

Errors: None.

See also: cycletorad (1021), degtograd (1022), radtodeg (1046), radtograd (1046), radtocycle (1045), gradtodeg (1026)

Listing: ./examples/mathex/ex16.pp

```
Program Example16;

{ Program to demonstrate the gradtorad function. }

Uses math;

begin
  writeln(gradtorad(100));
  writeln(gradtorad(200));
  writeln(gradtorad(300));
end.
```

63.13.35 Hypot

Synopsis: Return hypotenuse of triangle.

Declaration: `function Hypot(x: Float; y: Float) : Float`

Visibility: default

Description: `Hypot` returns the hypotenuse of the triangle where the sides adjacent to the square angle have lengths `x` and `y`. The function uses Pythagoras' rule for this.

Errors: None.

Listing: ./examples/mathex/ex17.pp

```
Program Example17;

{ Program to demonstrate the hypot function. }

Uses math;

begin
  Writeln(hypot(3,4)); // should be 5
end.
```

63.13.36 IfThen

Synopsis: Return one of two values, depending on a boolean condition.

Declaration:
```
function IfThen(val: Boolean; const iftrue: Integer;
                const iffalse: Integer) : Integer;  Overload
function IfThen(val: Boolean; const iftrue: Int64; const iffalse: Int64)
                 : Int64;  Overload
function IfThen(val: Boolean; const iftrue: Double;
                const iffalse: Double) : Double;  Overload
```

Visibility: default

Description: `ifthen` returns `iftrue` if `val` is `True`, and `iffalse` if `val` is `False`.

This function can be used in expressions.

63.13.37 InRange

Synopsis: Check whether value is in range.

Declaration:
```
function InRange(const AValue: Integer; const AMin: Integer;
                 const AMax: Integer) : Boolean;  Overload
function InRange(const AValue: Int64; const AMin: Int64;
                 const AMax: Int64) : Boolean;  Overload
function InRange(const AValue: Double; const AMin: Double;
                 const AMax: Double) : Boolean;  Overload
```

Visibility: default

Description: `InRange` returns `True` if `AValue` is in the range `AMin`..`AMax`. It returns `False` if `Value` lies outside the specified range.

See also: EnsureRange (1023)

63.13.38 InterestRate

Synopsis: Calculate the interest rate value of an investment.

Declaration:
```
function InterestRate(NPeriods: Integer; APayment: Float;
                      APresentValue: Float; AFutureValue: Float;
                      APaymentTime: TPaymentTime) : Float
```

Visibility: default

Description: `InterestRate` calculates the future value of an investment (`q`) in the cash flow formula (see CashFlowFunctions (1008)). The function result is the interest rate value in case of a future value `AFutureValue` for an investment of a start value `APresentValue` (`PV`), where `APayment` (`PMT`) is invested for `NPeriods` (`n`).

The `APaymentTime` parameter determines whether the investment (payment) is an ordinary annuity or an annuity due: `ptEndOfPeriod` must be used if payments are at the end of each period. If the payments are at the beginning of the periode, `ptStartOfPeriod` must be used.

See also: FutureValue (1025), NumberOfPeriods (1041), Payment (1042), PresentValue (1045), TPaymentTime (1013), CashFlowFunctions (1008)

63.13.39 IntPower

Synopsis: Return integer power.

Declaration:
```
function IntPower(base: Float; const exponent: Integer) : Float
```

Visibility: default

Description: `Intpower` returns `base` to the power `exponent`, where exponent is an integer value.

Errors: If `base` is zero and the exponent is negative, then an overflow error will occur.

See also: power (1044)

Listing: ./examples/mathex/ex18.pp

```
Program Example18;

{ Program to demonstrate the intpower function. }

Uses math;

Procedure DoIntpower (X : extended; Pow : Integer);

begin
  writeln(X:8:4,'^',Pow:2,' = ',intpower(X,pow):8:4);
end;

begin
  dointpower(0.0,0);
  dointpower(1.0,0);
  dointpower(2.0,5);
  dointpower(4.0,3);
  dointpower(2.0,-1);
  dointpower(2.0,-2);
  dointpower(-2.0,4);
  dointpower(-4.0,3);
end.
```

63.13.40 IsInfinite

Synopsis: Check whether value is infinite.

Declaration:
```
function IsInfinite(const d: Single) : Boolean;  Overload
function IsInfinite(const d: Double) : Boolean;  Overload
function IsInfinite(const d: Extended) : Boolean;  Overload
```

Visibility: default

Description: `IsInfinite` returns `True` if the double `d` contains the infinite value.

See also: IsZero (1030), IsInfinite (1029)

63.13.41 IsNan

Synopsis: Check whether value is Not a Number.

Declaration:
```
function IsNan(const d: Single) : Boolean;  Overload
function IsNan(const d: Double) : Boolean;  Overload
function IsNan(const d: Extended) : Boolean;  Overload
```

Visibility: default

Description: `IsNan` returns `True` if the double `d` contains Not A Number (a value which cannot be represented correctly in double format).

See also: IsZero (1030), IsInfinite (1029)

63.13.42 IsZero

Synopsis: Check whether value is zero.

Declaration:
```
function IsZero(const A: Single; Epsilon: Single) : Boolean;  Overload
function IsZero(const A: Single) : Boolean;  Overload
function IsZero(const A: Double; Epsilon: Double) : Boolean;  Overload
function IsZero(const A: Double) : Boolean;  Overload
function IsZero(const A: Extended; Epsilon: Extended) : Boolean
                ;  Overload
function IsZero(const A: Extended) : Boolean;  Overload
```

Visibility: default

Description: `IsZero` checks whether the float value `A` is zero, up to a precision of `Epsilon`. It returns `True` if Abs(`A`) is less than `Epsilon`.

The default value for `Epsilon` depends on the type of the argument: it is 1E-4 for Single, 1E-12 for Double and 1E-16 for extended.

See also: IsNan (1029), IsInfinite (1029), SameValue (1048)

63.13.43 Ldexp

Synopsis: Return (2 to the power p) times x.

Declaration:
```
function Ldexp(x: Float; const p: Integer) : Float
```

Visibility: default

Description: `Ldexp` returns (2 to the power `p`) times `x`.

Errors: None.

See also: lnxp1 (1031), log10 (1031), log2 (1032), logn (1032)

Listing: ./examples/mathex/ex19.pp

```
Program Example19;

{ Program to demonstrate the ldexp function. }

Uses math;

begin
  writeln(ldexp(2,4):8:4);
  writeln(ldexp(0.5,3):8:4);
end.
```

63.13.44 LnXP1

Synopsis: Return natural logarithm of 1+X.

Declaration: `function LnXP1(x: Float) : Float`

Visibility: default

Description: `Lnxp1` returns the natural logarithm of `1+X`. The result is more precise for small values of `x`. `x` should be larger than -1.

Errors: If $x\leq -1$ then an `EInvalidArgument` exception will be raised.

See also: ldexp (1030), log10 (1031), log2 (1032), logn (1032)

Listing: ./examples/mathex/ex20.pp

```
Program Example20;

{ Program to demonstrate the lnxp1 function. }

Uses math;

begin
  writeln(lnxp1(0));
  writeln(lnxp1(0.5));
  writeln(lnxp1(1));
end.
```

63.13.45 Log10

Synopsis: Return 10-Based logarithm.

Declaration: `function Log10(x: Float) : Float`

Visibility: default

Description: `Log10` returns the 10-base logarithm of `X`.

Errors: If `x` is less than or equal to 0 an 'invalid fpu operation' error will occur.

See also: ldexp (1030), lnxp1 (1031), log2 (1032), logn (1032)

Listing: ./examples/mathex/ex21.pp

```
Program Example21;

{ Program to demonstrate the log10 function. }

Uses math;

begin
  Writeln(Log10(10):8:4);
  Writeln(Log10(100):8:4);
  Writeln(Log10(1000):8:4);
  Writeln(Log10(1):8:4);
  Writeln(Log10(0.1):8:4);
  Writeln(Log10(0.01):8:4);
  Writeln(Log10(0.001):8:4);
end.
```

63.13.46 Log2

Synopsis: Return 2-based logarithm.

Declaration: `function Log2(x: Float) : Float`

Visibility: default

Description: `Log2` returns the 2-base logarithm of `X`.

Errors: If `x` is less than or equal to 0 an 'invalid fpu operation' error will occur.

See also: ldexp (1030), lnxp1 (1031), log10 (1031), logn (1032)

Listing: ./examples/mathex/ex22.pp

```
Program Example22;

{ Program to demonstrate the log2 function. }

Uses math;

begin
  Writeln(Log2(2):8:4);
  Writeln(Log2(4):8:4);
  Writeln(Log2(8):8:4);
  Writeln(Log2(1):8:4);
  Writeln(Log2(0.5):8:4);
  Writeln(Log2(0.25):8:4);
  Writeln(Log2(0.125):8:4);
end.
```

63.13.47 LogN

Synopsis: Return N-based logarithm.

Declaration: `function LogN(n: Float; x: Float) : Float`

Visibility: default

Description: `Logn` returns the n-base logarithm of `X`.

Errors: If `x` is less than or equal to 0 an 'invalid fpu operation' error will occur.

See also: ldexp (1030), lnxp1 (1031), log10 (1031), log2 (1032)

Listing: ./examples/mathex/ex23.pp

```
Program Example23;

{ Program to demonstrate the logn function. }

Uses math;

begin
  Writeln(Logn(3,4):8:4);
  Writeln(Logn(2,4):8:4);
  Writeln(Logn(6,9):8:4);
  Writeln(Logn(exp(1),exp(1)):8:4);
```

```
  WriteLn(Logn(0.5,1):8:4);
  WriteLn(Logn(0.25,3):8:4);
  WriteLn(Logn(0.125,5):8:4);
end.
```

63.13.48 Max

Synopsis: Returns the largest of two values.

Declaration:
```
function Max(a: Integer; b: Integer) : Integer;  Overload
function Max(a: Int64; b: Int64) : Int64;  Overload
function Max(a: QWord; b: QWord) : QWord;  Overload
function Max(a: Single; b: Single) : Single;  Overload
function Max(a: Double; b: Double) : Double;  Overload
function Max(a: Extended; b: Extended) : Extended;  Overload
```

Visibility: default

Description: `Max` returns the largest of the two values in the `a` and `b` arguments. The overloaded routines use the same type for both of the compared arguments. The type for the first argument determines the overloaded function used for the comparison.

Errors: None.

See also: min (1037), maxIntValue (1033), maxvalue (1034)

Listing: ./examples/mathex/ex24.pp

```
Program Example24;

{ Program to demonstrate the max function. }

Uses math;

Var
  A,B : Cardinal;

begin
  A:=1;b:=2;
  writeln(max(a,b));
end.
```

63.13.49 MaxIntValue

Synopsis: Return largest element in integer array.

Declaration:
```
function MaxIntValue(const Data: Array of Integer) : Integer
```

Visibility: default

Description: `MaxIntValue` returns the largest integer out of the `Data` array.

This function is provided for Delphi compatibility, use the maxvalue (1034) function instead.

Errors: None.

See also: maxvalue (1034), minvalue (1038), minIntValue (1038)

Listing: ./examples/mathex/ex25.pp

```
Program Example25;

{ Program to demonstrate the MaxIntValue function. }

{ Make sore integer is 32 bit}
{$mode objfpc}

Uses math;

Type
  TExArray = Array[1..100] of Integer;

Var
  I : Integer;
  ExArray : TExArray;

begin
  Randomize;
  for I:=low(exarray) to high(exarray) do
    ExArray[i]:=Random(I)-Random(100);
  Writeln(MaxIntValue(ExArray));
end.
```

63.13.50 MaxValue

Synopsis: Return largest value in array.

Declaration:
```
function MaxValue(const data: Array of Single) : Single
function MaxValue(const data: PSingle; const N: Integer) : Single
function MaxValue(const data: Array of Double) : Double
function MaxValue(const data: PDouble; const N: Integer) : Double
function MaxValue(const data: Array of Extended) : Extended
function MaxValue(const data: PExtended; const N: Integer) : Extended
function MaxValue(const data: Array of Integer) : Integer
function MaxValue(const data: PInteger; const N: Integer) : Integer
```

Visibility: default

Description: `Maxvalue` returns the largest value in the `data` array with integer or float values. The return value has the same type as the elements of the array.

The third and fourth forms accept a pointer to an array of `N` integer or float values.

Errors: None.

See also: maxIntValue (1033), minvalue (1038), minIntValue (1038)

Listing: ./examples/mathex/ex26.pp

```
program Example26;

{ Program to demonstrate the MaxValue function. }

{ Make sore integer is 32 bit}
{$mode objfpc}
```

```
uses math;

var i:1..100;
    f_array:array[1..100] of Float;
    i_array:array[1..100] of Integer;
    Pf_array:Pfloat;
    Pi_array:Pinteger;

begin
  randomize;

  Pf_array:=@f_array[1];
  Pi_array:=@i_array[1];

  for i:=low(f_array) to high(f_array) do
    f_array[i]:=(random-random)*100;
  for i:=low(i_array) to high(i_array) do
    i_array[i]:=random(l)-random(100);

  Writeln('Max Float       : ',MaxValue(f_array):8:4);
  Writeln('Max Float   (b) : ',MaxValue(Pf_array,100):8:4);
  Writeln('Max Integer     : ',MaxValue(i_array):8);
  Writeln('Max Integer (b) : ',MaxValue(Pi_array,100):8);
end.
```

63.13.51 Mean

Synopsis: Return mean value of array.

Declaration:
```
function Mean(const data: Array of Single) : Float
function Mean(const data: PSingle; const N: LongInt) : Float
function Mean(const data: Array of Double) : Float
function Mean(const data: PDouble; const N: LongInt) : Float
function Mean(const data: Array of Extended) : Float
function Mean(const data: PExtended; const N: LongInt) : Float
function Mean(const data: PInt64; const N: LongInt) : Float
function Mean(const data: Array of Int64) : Float
function Mean(const data: PInteger; const N: LongInt) : Float
function Mean(const data: Array of Integer) : Float
```

Visibility: default

Description: `Mean` returns the average value of `data`. The second form accepts a pointer to an array of `N` values.

Errors: None.

See also: meanandstddev (1036), momentskewkurtosis (1040), sum (1053)

Listing: ./examples/mathex/ex27.pp

```
Program Example27;

{ Program to demonstrate the Mean function. }
{ @ should return typed pointer }
{$T+}
Uses math;
```

```
Type
  TExArray = Array[1..100] of Float;

Var
  I : Integer;
  ExArray : TExArray;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  Writeln('Max       : ',MaxValue(ExArray):8:4);
  Writeln('Min       : ',MinValue(ExArray):8:4);
  Writeln('Mean      : ',Mean(ExArray):8:4);
  Writeln('Mean (b)  : ',Mean(PExtended(@ExArray[1]),100):8:4);
end.
```

63.13.52 MeanAndStdDev

Synopsis: Return mean and standard deviation of array.

Declaration:
```
procedure MeanAndStdDev(const data: Array of Single; var mean: Float;
                        var stddev: Float)
procedure MeanAndStdDev(const data: PSingle; const N: LongInt;
                        var mean: Float; var stddev: Float)
procedure MeanAndStdDev(const data: Array of Double; var mean: Float;
                        var stddev: Float)
procedure MeanAndStdDev(const data: PDouble; const N: LongInt;
                        var mean: Float; var stddev: Float)
procedure MeanAndStdDev(const data: Array of Extended; var mean: Float;
                        var stddev: Float)
procedure MeanAndStdDev(const data: PExtended; const N: LongInt;
                        var mean: Float; var stddev: Float)
```

Visibility: default

Description: `meanandstddev` calculates the mean and standard deviation of `data` and returns the result in `mean` and `stddev`, respectively. Stddev is zero if there is only one value. The second form accepts a pointer to an array of `N` values.

Errors: None.

See also: mean (1035), sum (1053), sumofsquares (1054), momentskewkurtosis (1040)

Listing: ./examples/mathex/ex28.pp

```
Program Example28;

{ Program to demonstrate the Meanandstddev function. }
{ @ should return typed pointer }
{$T+}

Uses math;

Type
  TExArray = Array[1..100] of Extended;
```

```
Var
  I : Integer;
  ExArray : TExArray;
  Mean,stddev : Extended;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  MeanAndStdDev(ExArray,Mean,StdDev);
  Writeln('Mean      : ',Mean:8:4);
  Writeln('StdDev    : ',StdDev:8:4);
  MeanAndStdDev(@ExArray[1],100,Mean,StdDev);
  Writeln('Mean  (b) : ',Mean:8:4);
  Writeln('StdDev (b) : ',StdDev:8:4);
end.
```

63.13.53 Min

Synopsis: Return smallest of two values.

Declaration:
```
function Min(a: Integer; b: Integer) : Integer;  Overload
function Min(a: Int64; b: Int64) : Int64;  Overload
function Min(a: QWord; b: QWord) : QWord;  Overload
function Min(a: Single; b: Single) : Single;  Overload
function Min(a: Double; b: Double) : Double;  Overload
function Min(a: Extended; b: Extended) : Extended;  Overload
```

Visibility: default

Description: `Min` returns the smallest of the two values in the `a` and `b` arguments. The overloaded variants use the same type for both of the compared arguments. The type for the first argument determines the overloaded function used for the comparison.

Errors: None.

See also: max (1033), minIntValue (1038), minValue (1038)

Listing: ./examples/mathex/ex29.pp

```
Program Example29;

{ Program to demonstrate the min function. }

Uses math;

Var
  A,B : Cardinal;

begin
  A:=1;b:=2;
  writeln(min(a,b));
end.
```

63.13.54 MinIntValue

Synopsis: Return smallest value in integer array.

Declaration:
```
function MinIntValue(const Data: Array of Integer) : Integer
```

Visibility: default

Description: `MinIntvalue` returns the smallest value in the `Data` array.

This function is provided for Delphi compatibility, use `minvalue` instead.

Errors: None

See also: minvalue (1038), maxIntValue (1033), maxvalue (1034)

Listing: ./examples/mathex/ex30.pp

```
Program Example30;

{ Program to demonstrate the MinIntValue function. }

{ Make sore integer is 32 bit}
{$mode objfpc}

Uses math;

Type
  TExArray = Array[1..100] of Integer;

Var
  I : Integer;
  ExArray : TExArray;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=Random(I)-Random(100);
  Writeln(MinIntValue(ExArray));
end.
```

63.13.55 MinValue

Synopsis: Return smallest value in array.

Declaration:
```
function MinValue(const data: Array of Single) : Single
function MinValue(const data: PSingle; const N: Integer) : Single
function MinValue(const data: Array of Double) : Double
function MinValue(const data: PDouble; const N: Integer) : Double
function MinValue(const data: Array of Extended) : Extended
function MinValue(const data: PExtended; const N: Integer) : Extended
function MinValue(const data: Array of Integer) : Integer
function MinValue(const Data: PInteger; const N: Integer) : Integer
```

Visibility: default

Description: `Minvalue` returns the smallest value in the `data` array with integer or float values. The return value has the same type as the elements of the array.

The third and fourth forms accept a pointer to an array of `N` integer or float values.

Errors: None.

See also: maxIntValue (1033), maxvalue (1034), minIntValue (1038)

Listing: ./examples/mathex/ex31.pp

```
program Example31;

{ Program to demonstrate the MinValue function. }

{ Make sure integer is 32 bit}
{$mode objfpc}

uses math;

var i:1..100;
    f_array:array[1..100] of Float;
    i_array:array[1..100] of Integer;
    Pf_array:Pfloat;
    PI_array:Pinteger;

begin
  randomize;

  Pf_array:=@f_array[1];
  Pi_array:=@i_array[1];

  for i:=low(f_array) to high(f_array) do
    f_array[i]:=(random-random)*100;
  for i:=low(i_array) to high(i_array) do
    i_array[i]:=random(l)-random(100);

  Writeln('Min Float       : ',MinValue(f_array):8:4);
  Writeln('Min Float   (b) : ',MinValue(Pf_array,100):8:4);
  Writeln('Min Integer     : ',MinValue(i_array):8);
  Writeln('Min Integer (b) : ',MinValue(Pi_array,100):8);
end.
```

63.13.56 modulus(Float,Float):Float

Synopsis: Floating point modulo.

Declaration: `operator mod(const a: Float; const b: Float) : Float`

Visibility: default

Description: `Modulus` is the floating-point equivalent of the modulo operation `a mod b`. It returns the result of

```
a-b * Int(a/b)
```

Errors: `b` may not be zero, but no check is performed.

See also: FMod (1024)

63.13.57 MomentSkewKurtosis

Synopsis: Return 4 first moments of distribution.

Declaration:
```
procedure MomentSkewKurtosis(const data: Array of Single;
                             out m1: Float; out m2: Float;
                             out m3: Float; out m4: Float;
                             out skew: Float; out kurtosis: Float)
procedure MomentSkewKurtosis(const data: PSingle; const N: Integer;
                             out m1: Float; out m2: Float;
                             out m3: Float; out m4: Float;
                             out skew: Float; out kurtosis: Float)
procedure MomentSkewKurtosis(const data: Array of Double;
                             out m1: Float; out m2: Float;
                             out m3: Float; out m4: Float;
                             out skew: Float; out kurtosis: Float)
procedure MomentSkewKurtosis(const data: PDouble; const N: Integer;
                             out m1: Float; out m2: Float;
                             out m3: Float; out m4: Float;
                             out skew: Float; out kurtosis: Float)
procedure MomentSkewKurtosis(const data: Array of Extended;
                             out m1: Float; out m2: Float;
                             out m3: Float; out m4: Float;
                             out skew: Float; out kurtosis: Float)
procedure MomentSkewKurtosis(const data: PExtended; const N: Integer;
                             out m1: Float; out m2: Float;
                             out m3: Float; out m4: Float;
                             out skew: Float; out kurtosis: Float)
```

Visibility: default

Description: `momentskewkurtosis` calculates the 4 first moments of the distribution of value in `data` and returns them in `m1`,`m2`,`m3` and `m4`, as well as the `skew` and `kurtosis`.

Errors: None.

See also: mean (1035), meanandstddev (1036)

Listing: ./examples/mathex/ex32.pp

```
program Example32;

{ Program to demonstrate the momentskewkurtosis function. }

uses math;

var distarray:array[1..1000] of float;
    l:longint;
    m1,m2,m3,m4,skew,kurtosis:float;

begin
  randomize;
  for l:=low(distarray) to high(distarray) do
    distarray[i]:=random;
  momentskewkurtosis(DistArray,m1,m2,m3,m4,skew,kurtosis);

  Writeln ('1st moment : ',m1:8:6);
  Writeln ('2nd moment : ',m2:8:6);
```

```
  WriteLn ('3rd moment : ',m3:8:6);
  WriteLn ('4th moment : ',m4:8:6);
  WriteLn ('Skew       : ',skew:8:6);
  WriteLn ('kurtosis   : ',kurtosis:8:6);
end.
```

63.13.58 Norm

Synopsis: Return Euclidean norm.

Declaration:
```
function Norm(const data: Array of Single) : Float
function Norm(const data: PSingle; const N: Integer) : Float
function Norm(const data: Array of Double) : Float
function Norm(const data: PDouble; const N: Integer) : Float
function Norm(const data: Array of Extended) : Float
function Norm(const data: PExtended; const N: Integer) : Float
```

Visibility: default

Description: `Norm` calculates the Euclidean norm of the array of data. This equals `sqrt(sumofsquares(data))`.

The second form accepts a pointer to an array of `N` values.

Errors: None.

See also: sumofsquares (1054)

Listing: ./examples/mathex/ex33.pp

```
program Example33;

{ Program to demonstrate the norm function. }

uses math;

var v:array[1..10] of Float;
    I:1..10;

begin
  for I:=low(v) to high(v) do
    v[i]:=random;
  writeln(norm(v));
end.
```

63.13.59 NumberOfPeriods

Synopsis: Calculate the number of periods for an investment.

Declaration:
```
function NumberOfPeriods(ARate: Float; APayment: Float;
                         APresentValue: Float; AFutureValue: Float;
                         APaymentTime: TPaymentTime) : Float
```

Visibility: default

Description: `NumberOfPeriods` calculates the number of periods (`n`) needed to obtain future value of an investment in the cash flow formula (see CashFlowFunctions (1008)). The function result is the number of periods a payment `APayment` (`PMT`) must be paid in order to obtain a future value `AFutureValue` for an investment of a start value `APresentValue` (`PV`), where `APayment` (`PMT`) is invested at a rate `ARate` (`q`).

The `APaymentTime` parameter determines whether the investment (payment) is an ordinary annuity or an annuity due: `ptEndOfPeriod` NumberOfPeriods must be used if payments are at the end of each period. If the payments are at the beginning of the periode, `ptStartOfPeriod` must be used.

See also: FutureValue (1025), InterestRate (1028), Payment (1042), PresentValue (1045), TPaymentTime (1013), CashFlowFunctions (1008)

63.13.60 Payment

Synopsis: Calculate the payment for an investment.

Declaration:
```
function Payment(ARate: Float; NPeriods: Integer; APresentValue: Float;
                 AFutureValue: Float; APaymentTime: TPaymentTime) : Float
```

Visibility: default

Description: `Payment` calculates the amount that must be payed (`PMT`) during number of periods (`n`) needed to obtain future value of an investment in the cash flow formula (see CashFlowFunctions (1008)). The function result is the amount (`PMT`) that must be paid in order to obtain a future value `AFutureValue` for an investment of a start value `APresentValue` (`PV`), where the amount must be payed `NPeriods` (`PMT`) and the interest rate is`ARate` (`q`).

The `APaymentTime` parameter determines whether the investment (payment) is an ordinary annuity or an annuity due: `ptEndOfPeriod` NumberOfPeriods must be used if payments are at the end of each period. If the payments are at the beginning of the periode, `ptStartOfPeriod` must be used.

See also: FutureValue (1025), InterestRate (1028), NumberOfPeriods (1041), PresentValue (1045), TPaymentTime (1013), CashFlowFunctions (1008)

63.13.61 PopnStdDev

Synopsis: Return Population standard deviation.

Declaration:
```
function PopnStdDev(const data: Array of Single) : Float
function PopnStdDev(const data: PSingle; const N: Integer) : Float
function PopnStdDev(const data: Array of Double) : Float
function PopnStdDev(const data: PDouble; const N: Integer) : Float
function PopnStdDev(const data: Array of Extended) : Float
function PopnStdDev(const data: PExtended; const N: Integer) : Float
```

Visibility: default

Description: `Popnstddev` returns the square root of the population variance of the values in the `Data` array. It returns zero if there is only one value.

The second form of this function accepts a pointer to an array of `N` values.

Errors: None.

See also: popnvariance (1043), mean (1035), meanandstddev (1036), stddev (1052), momentskewkurtosis (1040)

Listing: ./examples/mathex/ex35.pp

```
Program Example35;

{ Program to demonstrate the PopnStdDev function. }
{ @ should return typed pointer }
{$T+}

Uses Math;

Type
  TExArray = Array[1..100] of Float;

Var
  I : Integer;
  ExArray : TExArray;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  Writeln('Max               : ',MaxValue(ExArray):8:4);
  Writeln('Min               : ',MinValue(ExArray):8:4);
  Writeln('Pop. stddev.      : ',PopnStdDev(ExArray):8:4);
  Writeln('Pop. stddev. (b) : ',PopnStdDev(@ExArray[1],100):8:4);
end.
```

63.13.62 PopnVariance

Synopsis: Return population variance.

Declaration:
```
function PopnVariance(const data: PSingle; const N: Integer) : Float
function PopnVariance(const data: Array of Single) : Float
function PopnVariance(const data: PDouble; const N: Integer) : Float
function PopnVariance(const data: Array of Double) : Float
function PopnVariance(const data: PExtended; const N: Integer) : Float
function PopnVariance(const data: Array of Extended) : Float
```

Visibility: default

Description: `Popnvariance` the population variance of the values in the `Data` array. It returns zero if there is only one value.

The second form of this function accepts a pointer to an array of `N` values.

Errors: None.

See also: popnstddev (1042), mean (1035), meanandstddev (1036), stddev (1052), momentskewkurtosis (1040)

Listing: ./examples/mathex/ex36.pp

```
Program Example36;

{ Program to demonstrate the PopnVariance function. }
{ @ should return typed pointer }
{$T+}

Uses math;
```

```
Var
  I : Integer;
  ExArray : Array[1..100] of Float;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  Writeln('Max            : ',MaxValue(ExArray):8:4);
  Writeln('Min            : ',MinValue(ExArray):8:4);
  Writeln('Pop. var.      : ',PopnVariance(ExArray):8:4);
  Writeln('Pop. var. (b) : ',PopnVariance(@ExArray[1],100):8:4);
end.
```

63.13.63 Power

Synopsis: Return real power.

Declaration: `function Power(base: Float; exponent: Float) : Float`

Visibility: default

Description: `power` raises `base` to the power of `exponent`. This is equivalent to `exp(ln(base)*exponent)`. However, for integer `exponent` values intpower (1029) is called. Otherwise `base` must be non-negative.

Errors: Any number of floating point math exceptions may be raised, if they are not masked by SetExceptionMask (1049).

See also: intpower (1029)

Listing: ./examples/mathex/ex34.pp

```
Program Example34;

{ Program to demonstrate the power function. }

Uses Math;

procedure dopower(x,y : float);

begin
  writeln(x:8:6,'^',y:8:6,' = ',power(x,y):8:6)
end;

begin
  dopower(2,2);
  dopower(2,-2);
  dopower(2,0.0);
end.
```

63.13.64 power(Float,Float):Float

Synopsis: Raise base to the power exponent.

Declaration: `operator **(bas: Float; expo: Float) : Float`

Visibility: default

Description: `Power` raises `base` to the power `exponent`, i.e., it calculates `baseêxponent`.

63.13.65 power(Int64,Int64):Int64

Synopsis: Raise base to the power exponent.

Declaration: `operator **(bas: Int64; expo: Int64) : Int64`

Visibility: default

Description: `Power` raises `base` to the power `exponent`, i.e., it calculates `baseêxponent`.

63.13.66 PresentValue

Synopsis: Calculate the present value given the future value of an investment.

Declaration:
```
function PresentValue(ARate: Float; NPeriods: Integer; APayment: Float;
                      AFutureValue: Float; APaymentTime: TPaymentTime)
                       : Float
```

Visibility: default

Description: `PresentValue` calculates the present (start) value of an investment (`PV`) in the cash flow formula (see CashFlowFunctions (1008)) The function result is the start value of an investment, when the future value is `AFutureValue` (`FV`) and `APayment` (`PMT`) is invested for `NPeriods` (`n`) at the rate of `ARate` (`q`) per period.

The `APaymentTime` parameter determines whether the investment (payment) is an ordinary annuity or an annuity due: `ptEndOfPeriod` must be used if payments are at the end of each period. If the payments are at the beginning of the periode, `ptStartOfPeriod` must be used.

See also: FutureValue (1025), InterestRate (1028), NumberOfPeriods (1041), Payment (1042), TPaymentTime (1013), CashFlowFunctions (1008)

63.13.67 RadToCycle

Synopsis: Convert radians angle to cycle angle.

Declaration: `function RadToCycle(rad: Float) : Float`

Visibility: default

Description: `Radtocycle` converts its argument `rad` (an angle expressed in radians) to an angle in cycles.

(1 cycle equals 2 `pi` radians)

Errors: None.

See also: degtograd (1022), degtorad (1022), radtodeg (1046), radtograd (1046), cycletorad (1021)

Listing: ./examples/mathex/ex37.pp

```
Program Example37;

{ Program to demonstrate the radtocycle function. }

Uses math;

begin
  writeln(radtocycle(2*pi):8:6);
  writeln(radtocycle(pi):8:6);
  writeln(radtocycle(pi/2):8:6);
end.
```

63.13.68 RadToDeg

Synopsis: Convert radians angle to degrees angle.

Declaration: `function RadToDeg(rad: Float) : Float`

Visibility: default

Description: `Radtodeg` converts its argument `rad` (an angle expressed in radians) to an angle in degrees. (180 degrees equals pi radians)

Errors: None.

See also: degtograd (1022), degtorad (1022), radtocycle (1045), radtograd (1046), cycletorad (1021)

Listing: ./examples/mathex/ex38.pp

```
Program Example38;

{ Program to demonstrate the radtodeg function. }

Uses math;

begin
  writeln(radtodeg(2*pi):8:6);
  writeln(radtodeg(pi):8:6);
  writeln(radtodeg(pi/2):8:6);
end.
```

63.13.69 RadToGrad

Synopsis: Convert radians angle to grads angle.

Declaration: `function RadToGrad(rad: Float) : Float`

Visibility: default

Description: `Radtodeg` converts its argument `rad` (an angle expressed in radians) to an angle in grads.
(200 grads equals pi radians)

Errors: None.

See also: degtograd (1022), degtorad (1022), radtocycle (1045), radtodeg (1046), cycletorad (1021)

Listing: ./examples/mathex/ex39.pp

```
Program Example39;

{ Program to demonstrate the radtograd function. }

Uses math;

begin
  writeln(radtograd(2*pi):8:6);
  writeln(radtograd(pi):8:6);
  writeln(radtograd(pi/2):8:6);
end.
```

63.13.70 RandG

Synopsis: Return gaussian distributed random number.

Declaration: `function RandG(mean: Float; stddev: Float) : Float`

Visibility: default

Description: `randg` returns a random number which - when produced in large quantities - has a Gaussian distribution with mean `mean` and standarddeviation `stddev`.

Errors: None.

See also: mean (1035), stddev (1052), meanandstddev (1036)

Listing: ./examples/mathex/ex40.pp

```
Program Example40;

{ Program to demonstrate the randg function. }

Uses Math;

Var
  I : Integer;
  ExArray : Array[1..10000] of Float;
  Mean,stddev : Float;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=Randg(1,0.2);
  MeanAndStdDev(ExArray,Mean,StdDev);
  Writeln('Mean      : ',Mean:8:4);
  Writeln('StdDev    : ',StdDev:8:4);
end.
```

63.13.71 RandomFrom

Synopsis: Return a random element of an array of numbers.

Declaration:
```
function RandomFrom(const AValues: Array of Double) : Double;  Overload
function RandomFrom(const AValues: Array of Integer) : Integer
                    ;  Overload
function RandomFrom(const AValues: Array of Int64) : Int64;  Overload
```

Visibility: default

Description: `RandomFrom` returns a random element from the array `AValues`. The return value has the same type as the type of the array elements.

See also: #rtl.system.Random (1546), RandomRange (1048)

63.13.72 RandomRange

Synopsis: Return a random number in a range.

Declaration:
```
function RandomRange(const aFrom: Integer; const aTo: Integer) : Integer
function RandomRange(const aFrom: Int64; const aTo: Int64) : Int64
```

Visibility: default

Description: `RandomRange` returns a random value in the range `AFrom` to `ATo`. `AFrom` and `ATo` do not need to be in increasing order. The upper border is not included in the generated value, but the lower border is.

See also: #rtl.system.Random (1546), RandomFrom (1047)

63.13.73 RoundTo

Synopsis: Round to the specified number of digits.

Declaration:
```
function RoundTo(const AValue: Double; const Digits: TRoundToRange)
                 : Double
function RoundTo(const AVAlue: Extended; const Digits: TRoundToRange)
                 : Extended
function RoundTo(const AValue: Single; const Digits: TRoundToRange)
                 : Single
```

Visibility: default

Description: `RoundTo` rounds the specified float `AValue` to the specified number of digits and returns the result. This result is accurate to "10 to the power Digits". It uses the standard `Round` function for this.

Errors: An exception may occur if the value `AValue` is not inside a valid integer (or Int64) range.

See also: TRoundToRange (1013), SimpleRoundTo (1050), SetRoundMode (1050), TFPURoundingMode (1013)

63.13.74 SameValue

Synopsis: Check whether 2 float values are the same.

Declaration:
```
function SameValue(const A: Extended; const B: Extended) : Boolean
                   ;  Overload
function SameValue(const A: Double; const B: Double) : Boolean
                   ;  Overload
function SameValue(const A: Single; const B: Single) : Boolean
```

```
                    ;  Overload
function SameValue(const A: Extended; const B: Extended;
                   Epsilon: Extended) : Boolean;  Overload
function SameValue(const A: Double; const B: Double; Epsilon: Double)
                    : Boolean;  Overload
function SameValue(const A: Single; const B: Single; Epsilon: Single)
                    : Boolean;  Overload
```

Visibility: default

Description: `SameValue` returns `True` if the floating-point values `A` and `B` are the same, i.e. whether the absolute value of their their difference is smaller than `Epsilon`. If their difference is larger, then `False` is returned.

If unspecified, the default value for `Epsilon` is 0.0.

See also: MinFloat (1012), IsZero (1030)

63.13.75 Sec

Synopsis: Alias for `secant`.

Declaration: `function Sec(x: Float) : Float`

Visibility: default

Description: `sec` is an alias for the secant (1049) function.

See also: secant (1049)

63.13.76 Secant

Synopsis: Calculate secant.

Declaration: `function Secant(x: Float) : Float`

Visibility: default

Description: `Secant` calculates the secant (1/`cos(x)`) of its argument `x`.

Errors: If 90 or 270 degrees is specified, an exception will be raised.

See also: cosecant (1019)

63.13.77 SetExceptionMask

Synopsis: Set the Floating Point Unit exception mask.

Declaration: `function SetExceptionMask(const Mask: TFPUExceptionMask)`
`: TFPUExceptionMask`

Visibility: default

Description: Set the Floating Point Unit exception mask.

63.13.78 SetPrecisionMode

Synopsis: Set the Floating Point Unit precision mode.

Declaration:
```
function SetPrecisionMode(const Precision: TFPUPrecisionMode)
                          : TFPUPrecisionMode
```

Visibility: default

Description: Set the Floating Point Unit precision mode.

63.13.79 SetRoundMode

Synopsis: Set the Floating Point Unit rounding mode.

Declaration:
```
function SetRoundMode(const RoundMode: TFPURoundingMode)
                      : TFPURoundingMode
```

Visibility: default

Description: `SetRoundMode` sets the floating point unit rounding mode. It also returns the previous rounding mode.

rmNearestRound to nearest integer.

rmDownRound to biggest integer smaller than value.

rmUpRound to smallest integer larger than value.

rmTruncateCut off fractional part.

See also: GetRoundMode (1026), TFPURoundingMode (1013)

63.13.80 Sign

Synopsis: Return sign of argument.

Declaration:
```
function Sign(const AValue: Integer) : TValueSign;  Overload
function Sign(const AValue: Int64) : TValueSign;  Overload
function Sign(const AValue: Single) : TValueSign;  Overload
function Sign(const AValue: Double) : TValueSign;  Overload
function Sign(const AValue: Extended) : TValueSign;  Overload
```

Visibility: default

Description: `Sign` returns the sign of it's argument, which can be an Integer, 64 bit integer, or a double. The returned value is an integer which is -1, 0 or 1, and can be used to do further calculations with.

63.13.81 SimpleRoundTo

Synopsis: Round to the specified number of digits (rounding up if needed).

Declaration:
```
function SimpleRoundTo(const AValue: Single;
                       const Digits: TRoundToRange) : Single
function SimpleRoundTo(const AValue: Double;
                       const Digits: TRoundToRange) : Double
function SimpleRoundTo(const AValue: Extended;
                       const Digits: TRoundToRange) : Extended
```

Visibility: default

Description: `SimpleRoundTo` rounds the specified float `AValue` to the specified number of `digits` using symmetric arithmetic rounding (rounding up for positive or down for negative) and returns the result. This result is accurate to "10 to the power `Digits`". It uses the standard `Round` function for this.

Errors: An exception may occur if the value `AValue` is not inside a valid integer (or Int64) range.

See also: TRoundToRange (1013), RoundTo (1048)

63.13.82 SinCos

Synopsis: Return sine and cosine of argument.

Declaration:
```
procedure SinCos(theta: single; out sinus: single; out cosinus: single)
procedure SinCos(theta: Double; out sinus: Double; out cosinus: Double)
procedure SinCos(theta: extended; out sinus: extended;
                 out cosinus: extended)
```

Visibility: default

Description: `Sincos` calculates the sine and cosine of the angle `theta`, and returns the result in `sinus` and `cosinus`.

On Intel hardware, This calculation will be faster than making 2 calls to calculate the sine and cosine separately.

Errors: None.

See also: arcsin (1015), arccos (1014)

Listing: ./examples/mathex/ex41.pp

```
Program Example41;

{ Program to demonstrate the sincos function. }

Uses math;

Procedure dosincos(Angle : Float);

Var
  Sine,Cosine : Float;

begin
  sincos(angle,sine,cosine);
  Write('Angle : ',Angle:8:6);
  Write(' Sine :',sine:8:6);
  Write(' Cosine :',cosine:8:6);
end;

begin
  dosincos(pi);
  dosincos(pi/2);
  dosincos(pi/3);
  dosincos(pi/4);
  dosincos(pi/6);
end.
```

63.13.83 SinH

Synopsis: Return hyperbolic sine.

Declaration:
```
function SinH(x: Float) : Float
```

Visibility: default

Description: `Sinh` returns the hyperbolic sine of its argument `x`.

See also: cosh (1020), arsinh (1017), tanh (1056), artanh (1017)

Listing: ./examples/mathex/ex42.pp

```
Program Example42;

{ Program to demonstrate the sinh function. }

Uses math;

begin
  writeln(sinh(0));
  writeln(sinh(1));
  writeln(sinh(-1));
end.
```

63.13.84 StdDev

Synopsis: Return standard deviation of data.

Declaration:
```
function StdDev(const data: Array of Single) : Float
function StdDev(const data: PSingle; const N: Integer) : Float
function StdDev(const data: Array of Double) : Float
function StdDev(const data: PDouble; const N: Integer) : Float
function StdDev(const data: Array of Extended) : Float
function StdDev(const data: PExtended; const N: Integer) : Float
```

Visibility: default

Description: `Stddev` returns the standard deviation of the values in `Data`. It returns zero if there is only one value.

The second form of the function accepts a pointer to an array of `N` values.

Errors: None.

See also: mean (1035), meanandstddev (1036), variance (1058), totalvariance (1057)

Listing: ./examples/mathex/ex43.pp

```
Program Example40;

{ Program to demonstrate the stddev function. }
{ @ should return typed pointer }
{$T+}

Uses Math;

Var
```

```
  I : Integer;
  ExArray : Array[1..10000] of Float;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=Randg(1,0.2);
  Writeln('StdDev      : ',StdDev(ExArray):8:4);
  Writeln('StdDev (b) : ',StdDev(@ExArray[1],10000):8:4);
end.
```

63.13.85 Sum

Synopsis: Return sum of values.

Declaration:
```
function Sum(const data: Array of Single) : Float
function Sum(const data: PSingle; const N: LongInt) : Float
function Sum(const data: Array of Double) : Float
function Sum(const data: PDouble; const N: LongInt) : Float
function Sum(const data: Array of Extended) : Float
function Sum(const data: PExtended; const N: LongInt) : Float
```

Visibility: default

Description: `Sum` returns the sum of the values in the `data` array.

The second form of the function accepts a pointer to an array of `N` values.

Errors: None.

See also: sumofsquares (1054), sumsandsquares (1055), totalvariance (1057), variance (1058)

Listing: ./examples/mathex/ex44.pp

```
Program Example44;

{ Program to demonstrate the Sum function. }
{ @ should return typed pointer }
{$T+}

Uses math;

Var
  I : 1..100;
  ExArray : Array[1..100] of Float;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  Writeln('Max      : ',MaxValue(ExArray):8:4);
  Writeln('Min      : ',MinValue(ExArray):8:4);
  Writeln('Sum      : ',Sum(ExArray):8:4);
  Writeln('Sum (b) : ',Sum(@ExArray[1],100):8:4);
end.
```

63.13.86 SumInt

Synopsis: Return the sum of an array of integers.

Declaration:
```
function SumInt(const data: PInt64; const N: LongInt) : Int64
function SumInt(const data: Array of Int64) : Int64
function SumInt(const data: PInteger; const N: LongInt) : Int64
function SumInt(const data: Array of Integer) : Int64
```

Visibility: default

Description: `SumInt` returns the sum of the `N` integers in the `Data` array, where this can be an open array or a pointer to an array.

Errors: An overflow may occur.

63.13.87 SumOfSquares

Synopsis: Return sum of squares of values.

Declaration:
```
function SumOfSquares(const data: Array of Single) : Float
function SumOfSquares(const data: PSingle; const N: Integer) : Float
function SumOfSquares(const data: Array of Double) : Float
function SumOfSquares(const data: PDouble; const N: Integer) : Float
function SumOfSquares(const data: Array of Extended) : Float
function SumOfSquares(const data: PExtended; const N: Integer) : Float
```

Visibility: default

Description: `Sumofsquares` returns the sum of the squares of the values in the `data` array.

The second form of the function accepts a pointer to an array of `N` values.

Errors: None.

See also: sum (1053), sumsandsquares (1055), totalvariance (1057), variance (1058)

Listing: ./examples/mathex/ex45.pp

```
Program Example45;

{ Program to demonstrate the SumOfSquares function. }
{ @ should return typed pointer }
{$T+}

Uses math;

Var
  I : 1..100;
  ExArray : Array[1..100] of Float;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  Writeln('Max              : ',MaxValue(ExArray):8:4);
  Writeln('Min              : ',MinValue(ExArray):8:4);
  Writeln('Sum squares      : ',SumOfSquares(ExArray):8:4);
  Writeln('Sum squares (b)  : ',SumOfSquares(@ExArray[1],100):8:4);
end.
```

63.13.88 SumsAndSquares

Synopsis: Return sum and sum of squares of values.

Declaration:
```
procedure SumsAndSquares(const data: Array of Single; var sum: Float;
                         var sumofsquares: Float)
procedure SumsAndSquares(const data: PSingle; const N: Integer;
                         var sum: Float; var sumofsquares: Float)
procedure SumsAndSquares(const data: Array of Double; var sum: Float;
                         var sumofsquares: Float)
procedure SumsAndSquares(const data: PDouble; const N: Integer;
                         var sum: Float; var sumofsquares: Float)
procedure SumsAndSquares(const data: Array of Extended; var sum: Float;
                         var sumofsquares: Float)
procedure SumsAndSquares(const data: PExtended; const N: Integer;
                         var sum: Float; var sumofsquares: Float)
```

Visibility: default

Description: `sumsandsquares` calculates the sum of the values and the sum of the squares of the values in the `data` array and returns the results in `sum` and `sumofsquares`.

The second form of the function accepts a pointer to an array of `N` values.

Errors: None.

See also: sum (1053), sumofsquares (1054), totalvariance (1057), variance (1058)

Listing: ./examples/mathex/ex46.pp

```
Program Example45;

{ Program to demonstrate the SumOfSquares function. }
{ @ should return typed pointer }
{$T+}

Uses math;

Var
  I : 1..100;
  ExArray : Array[1..100] of Float;
  s,ss : float;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  Writeln('Max              : ',MaxValue(ExArray):8:4);
  Writeln('Min              : ',MinValue(ExArray):8:4);
  SumsAndSquares(ExArray,S,SS);
  Writeln('Sum              : ',S:8:4);
  Writeln('Sum squares      : ',SS:8:4);
  SumsAndSquares(@ExArray[1],100,S,SS);
  Writeln('Sum (b)          : ',S:8:4);
  Writeln('Sum squares (b)  : ',SS:8:4);
end.
```

63.13.89 Tan

Synopsis: Return tangent.

Declaration: `function Tan(x: Float) : Float`

Visibility: default

Description: `Tan` returns the tangent of `x`. The argument `x` must be in radians.

Errors: If `x` (normalized) is pi/2 or 3pi/2 then an overflow will occur.

See also: tanh (1056), arcsin (1015), sincos (1051), arccos (1014)

Listing: ./examples/mathex/ex47.pp

```
Program Example47;

{ Program to demonstrate the Tan function. }

Uses math;

Procedure DoTan(Angle : Float);

begin
  Write('Angle : ',RadToDeg(Angle):8:6);
  Writeln(' Tangent : ',Tan(Angle):8:6);
end;

begin
  DoTan(0);
  DoTan(Pi);
  DoTan(Pi/3);
  DoTAn(Pi/4);
  DoTan(Pi/6);
end.
```

63.13.90 TanH

Synopsis: Return hyperbolic tangent.

Declaration: `function TanH(x: Float) : Float`

Visibility: default

Description: `Tanh` returns the hyperbolic tangent of `x`.

Errors: None.

See also: arcsin (1015), sincos (1051), arccos (1014)

Listing: ./examples/mathex/ex48.pp

```
Program Example48;

{ Program to demonstrate the Tanh function. }

Uses math;
```

```
begin
  writeln(tanh(0));
  writeln(tanh(1));
  writeln(tanh(-1));
end.
```

63.13.91 TotalVariance

Synopsis: Return total variance of values.

Declaration:
```
function TotalVariance(const data: Array of Single) : Float
function TotalVariance(const data: PSingle; const N: Integer) : Float
function TotalVariance(const data: Array of Double) : Float
function TotalVariance(const data: PDouble; const N: Integer) : Float
function TotalVariance(const data: Array of Extended) : Float
function TotalVariance(const data: PExtended; const N: Integer) : Float
```

Visibility: default

Description: `TotalVariance` returns the total variance of the values in the `data` array. It returns zero if there is only one value.

The second form of the function accepts a pointer to an array of `N` values.

Errors: None.

See also: variance (1058), stddev (1052), mean (1035)

Listing: ./examples/mathex/ex49.pp

```
Program Example49;

{ Program to demonstrate the TotalVariance function. }
{ @ should return typed pointer }
{$T+}

Uses math;

Type
  TExArray = Array[1..100] of Float;

Var
  I : Integer;
  ExArray : TExArray;
  TV : float;

begin
  Randomize;
  for I:=1 to 100 do
    ExArray[i]:=(Random-Random)*100;
  TV:=TotalVariance(ExArray);
  Writeln('Total variance     : ',TV:8:4);
  TV:=TotalVariance(@ExArray[1],100);
  Writeln('Total Variance (b) : ',TV:8:4);
end.
```

63.13.92 Variance

Synopsis: Return variance of values.

Declaration:
```
function Variance(const data: Array of Single) : Float
function Variance(const data: PSingle; const N: Integer) : Float
function Variance(const data: Array of Double) : Float
function Variance(const data: PDouble; const N: Integer) : Float
function Variance(const data: Array of Extended) : Float
function Variance(const data: PExtended; const N: Integer) : Float
```

Visibility: default

Description: `Variance` returns the variance of the values in the `data` array. It returns zero if there is only one value.

The second form of the function accepts a pointer to an array of `N` values.

Errors: None.

See also: totalvariance (1057), stddev (1052), mean (1035)

Listing: ./examples/mathex/ex50.pp

```
Program Example50;

{ Program to demonstrate the Variance function. }
{ @ should return typed pointer }
{$T+}

Uses math;

Var
  I : 1..100;
  ExArray : Array[1..100] of Float;
  V : float;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  V:=Variance(ExArray);
  Writeln('Variance      : ',V:8:4);
  V:=Variance(@ExArray[1],100);
  Writeln('Variance (b) : ',V:8:4);
end.
```

Listing: ./examples/mathex/ex51.pp

```
Program Example51;

{
  Program to demonstrate the Variance function.
  It demonstrates the absence of large errors in the calculation.
}

Uses math;

const
  Size = 1000000;
```

```
var
  dataS: array of Single;
  dataD: array of Double;
  dataE: array of Extended;
  i,n: longint;

begin
  WriteLn('Each run should return a value near unity.');
  WriteLn('Single:');
  SetLength( dataS, Size );
  for n := 1 to 4 do
  begin
    for i := 0 to Size - 1 do
    begin
      dataS[i] := 10000000 + RandG(0,1);
    end;
    WriteLn( Math.Variance( dataS ):5:3 );
  end;

  WriteLn('Double:');
  SetLength( dataD, Size );
  for n := 1 to 4 do
  begin
    for i := 0 to Size - 1 do
    begin
      dataD[i] := 1000000000000000 + RandG(0,1);
    end;
    WriteLn( Math.Variance( dataD ):5:3 );
  end;

  WriteLn('Extended:');
  SetLength( dataE, Size );
  for n := 1 to 4 do
  begin
    for i := 0 to Size - 1 do
    begin
      dataE[i] := 1000000000000000000 + RandG(0,1);
    end;
    WriteLn( Math.Variance( dataE ):5:3 );
  end;
end.
```

63.14 EInvalidArgument

63.14.1 Description

Exception raised when invalid arguments are passed to a function.

Chapter 64

Reference for unit 'matrix'

64.1 Used units

Table 64.1: Used units by unit 'matrix'

Name	Page
System	1362

64.2 Overview

The unit matrix is a unit that provides objects for the common two, three and four dimensional vectors matrices. These vectors and matrices are very common in computer graphics and are often implemented from scratch by programmers while every implementation provides exactly the same functionality.

It makes therefore sense to provide this functionality in the runtime library. This eliminates the need for programmers to reinvent the wheel and also allows libraries that use matrix operations to become more compatible.

The matrix unit does not provide n-dimensional matrices. The functionality needs of a general matrix unit varies from application to application; one can think of reduced memory usage tricks for matrices that only have data around the diagonal etc., desire for parallelization etc. etc. It is believed that programmers that do use n-dimensional matrices would not necessarily benefit from such a unit in the runtime library.

Design goals:

- Provide common dimensions, two three and four.
- Provide multiple floating point precisions, single, double, extended.
- Simple trivial binary representation; it is possible to typecast vectors into other implementations that use the same trivial representation.
- No dynamic memory management in the background. It must be possible to write expressions like matrix A * B * C without worrying about memory management.

Design decisions:

- Class object model is ruled out. The objects object model, without virtual methods, is suitable.
- Operator overloading is a good way to allow programmers to write matrix expressions.
- 3 dimensions * 3 precision means 9 vector and 9 matrix objects. Macro's have been used in the source to take care of this.

64.3 Constants, types and variables

64.3.1 Types

```
Tmatrix2_double_data = Array[0..1,0..1] of Double
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix2_extended_data = Array[0..1,0..1] of extended
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix2_single_data = Array[0..1,0..1] of single
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix3_double_data = Array[0..2,0..2] of Double
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix3_extended_data = Array[0..2,0..2] of extended
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix3_single_data = Array[0..2,0..2] of single
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix4_double_data = Array[0..3,0..3] of Double
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix4_extended_data = Array[0..3,0..3] of extended
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix4_single_data = Array[0..3,0..3] of single
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector2_double_data = Array[0..1] of Double
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector2_extended_data = Array[0..1] of extended
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector2_single_data = Array[0..1] of single
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector3_double_data = Array[0..2] of Double
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector3_extended_data = Array[0..2] of extended
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector3_single_data = Array[0..2] of single
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector4_double_data = Array[0..3] of Double
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector4_extended_data = Array[0..3] of extended
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector4_single_data = Array[0..3] of single
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

64.4 Procedures and functions

64.4.1 add(Tmatrix2_double,Double):Tmatrix2_double

Synopsis: Add scalar to two-dimensional double precision matrix.

Declaration:
```
operator +(const m: Tmatrix2_double; const x: Double) : Tmatrix2_double
```

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.4.2 add(Tmatrix2_double,Tmatrix2_double):Tmatrix2_double

Synopsis: Add two two-dimensional double precision matrices together.

Declaration:
```
operator +(const m1: Tmatrix2_double; const m2: Tmatrix2_double)
                : Tmatrix2_double
```

Visibility: default

Description: This operator allows you to add two two-dimensional double precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.4.3 add(Tmatrix2_extended,extended):Tmatrix2_extended

Synopsis: Add scalar to two-dimensional extended precision matrix.

Declaration:
```
operator +(const m: Tmatrix2_extended; const x: extended)
                : Tmatrix2_extended
```

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.4.4 add(Tmatrix2_extended,Tmatrix2_extended):Tmatrix2_extended

Synopsis: Add two two-dimensional extended precision matrices together.

Declaration:
```
operator +(const m1: Tmatrix2_extended; const m2: Tmatrix2_extended)
                : Tmatrix2_extended
```

Visibility: default

Description: This operator allows you to add two two-dimensional extended precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.4.5 add(Tmatrix2_single,single):Tmatrix2_single

Synopsis: Add scalar to two-dimensional single precision matrix.

Declaration:
```
operator +(const m: Tmatrix2_single; const x: single) : Tmatrix2_single
```

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.4.6 add(Tmatrix2_single,Tmatrix2_single):Tmatrix2_single

Synopsis: Add two two-dimensional single precision matrices together.

Declaration:
```
operator +(const m1: Tmatrix2_single; const m2: Tmatrix2_single)
                : Tmatrix2_single
```

Visibility: default

Description: This operator allows you to add two two-dimensional single precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.4.7 add(Tmatrix3_double,Double):Tmatrix3_double

Synopsis: Add scalar to three-dimensional double precision matrix.

Declaration:
```
operator +(const m: Tmatrix3_double; const x: Double) : Tmatrix3_double
```

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.4.8 add(Tmatrix3_double,Tmatrix3_double):Tmatrix3_double

Synopsis: Add two three-dimensional double precision matrices together.

Declaration:
```
operator +(const m1: Tmatrix3_double; const m2: Tmatrix3_double)
                : Tmatrix3_double
```

Visibility: default

Description: This operator allows you to add two three-dimensional double precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.4.9 add(Tmatrix3_extended,extended):Tmatrix3_extended

Synopsis: Add scalar to three-dimensional extended precision matrix.

Declaration:
```
operator +(const m: Tmatrix3_extended; const x: extended)
                : Tmatrix3_extended
```

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.4.10 add(Tmatrix3_extended,Tmatrix3_extended):Tmatrix3_extended

Synopsis: Add two three-dimensional extended precision matrices together.

Declaration:
```
operator +(const m1: Tmatrix3_extended; const m2: Tmatrix3_extended)
                : Tmatrix3_extended
```

Visibility: default

Description: This operator allows you to add two three-dimensional extended precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.4.11 add(Tmatrix3_single,single):Tmatrix3_single

Synopsis: Add scalar to three-dimensional single precision matrix.

Declaration:
```
operator +(const m: Tmatrix3_single; const x: single) : Tmatrix3_single
```

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.4.12 add(Tmatrix3_single,Tmatrix3_single):Tmatrix3_single

Synopsis: Add two three-dimensional single precision matrices together.

Declaration:
```
operator +(const m1: Tmatrix3_single; const m2: Tmatrix3_single)
                : Tmatrix3_single
```

Visibility: default

Description: This operator allows you to add two three-dimensional single precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.4.13 add(Tmatrix4_double,Double):Tmatrix4_double

Synopsis: Add scalar to four-dimensional double precision matrix.

Declaration:
```
operator +(const m: Tmatrix4_double; const x: Double) : Tmatrix4_double
```

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.4.14 add(Tmatrix4_double,Tmatrix4_double):Tmatrix4_double

Synopsis: Add two four-dimensional double precision matrices together.

Declaration:
```
operator +(const m1: Tmatrix4_double; const m2: Tmatrix4_double)
                : Tmatrix4_double
```

Visibility: default

Description: This operator allows you to add two four-dimensional double precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.4.15 add(Tmatrix4_extended,extended):Tmatrix4_extended

Synopsis: Add scalar to four-dimensional extended precision matrix.

Declaration:
```
operator +(const m: Tmatrix4_extended; const x: extended)
                : Tmatrix4_extended
```

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.4.16 add(Tmatrix4_extended,Tmatrix4_extended):Tmatrix4_extended

Synopsis: Add two four-dimensional extended precision matrices together.

Declaration:
```
operator +(const m1: Tmatrix4_extended; const m2: Tmatrix4_extended)
                : Tmatrix4_extended
```

Visibility: default

Description: This operator allows you to add two four-dimensional extended precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.4.17 add(Tmatrix4_single,single):Tmatrix4_single

Synopsis: Add scalar to four-dimensional single precision matrix.

Declaration:
```
operator +(const m: Tmatrix4_single; const x: single) : Tmatrix4_single
```

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.4.18 add(Tmatrix4_single,Tmatrix4_single):Tmatrix4_single

Synopsis: Add two four-dimensional single precision matrices together.

Declaration:
```
operator +(const m1: Tmatrix4_single; const m2: Tmatrix4_single)
                : Tmatrix4_single
```

Visibility: default

Description: This operator allows you to add two four-dimensional single precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.4.19 add(Tvector2_double,Double):Tvector2_double

Synopsis: Add scalar to two-dimensional double precision vector.

Declaration:
```
operator +(const x: Tvector2_double; y: Double) : Tvector2_double
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.4.20 add(Tvector2_double,Tvector2_double):Tvector2_double

Synopsis: Add two-dimensional double precision vectors together.

Declaration:
```
operator +(const x: Tvector2_double; const y: Tvector2_double)
                : Tvector2_double
```

Visibility: default

Description: This operator allows you to add two two-dimensional vectors with double precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.4.21 add(Tvector2_extended,extended):Tvector2_extended

Synopsis: Add scalar to two-dimensional extended precision vector.

Declaration:
```
operator +(const x: Tvector2_extended; y: extended) : Tvector2_extended
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.4.22 add(Tvector2_extended,Tvector2_extended):Tvector2_extended

Synopsis: Add two-dimensional extended precision vectors together.

Declaration:
```
operator +(const x: Tvector2_extended; const y: Tvector2_extended)
                : Tvector2_extended
```

Visibility: default

Description: This operator allows you to add two two-dimensional vectors with extended precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.4.23 add(Tvector2_single,single):Tvector2_single

Synopsis: Add scalar to two-dimensional single precision vector.

Declaration:
```
operator +(const x: Tvector2_single; y: single) : Tvector2_single
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.4.24 add(Tvector2_single,Tvector2_single):Tvector2_single

Synopsis: Add two-dimensional single precision vectors together.

Declaration:
```
operator +(const x: Tvector2_single; const y: Tvector2_single)
                : Tvector2_single
```

Visibility: default

Description: This operator allows you to add two two-dimensional vectors with single precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.4.25 add(Tvector3_double,Double):Tvector3_double

Synopsis: Add scalar to three-dimensional double precision vector.

Declaration:
```
operator +(const x: Tvector3_double; y: Double) : Tvector3_double
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.4.26 add(Tvector3_double,Tvector3_double):Tvector3_double

Synopsis: Add three-dimensional double precision vectors together.

Declaration:
```
operator +(const x: Tvector3_double; const y: Tvector3_double)
                : Tvector3_double
```

Visibility: default

Description: This operator allows you to add two three-dimensional vectors with double precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.4.27 add(Tvector3_extended,extended):Tvector3_extended

Synopsis: Add scalar to three-dimensional extended precision vector.

Declaration:
```
operator +(const x: Tvector3_extended; y: extended) : Tvector3_extended
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.4.28 add(Tvector3_extended,Tvector3_extended):Tvector3_extended

Synopsis: Add three-dimensional extended precision vectors together.

Declaration:
```
operator +(const x: Tvector3_extended; const y: Tvector3_extended)
                : Tvector3_extended
```

Visibility: default

Description: This operator allows you to add two three-dimensional vectors with extended precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.4.29 add(Tvector3_single,single):Tvector3_single

Synopsis: Add scalar to three-dimensional single precision vector.

Declaration:
```
operator +(const x: Tvector3_single; y: single) : Tvector3_single
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.4.30 add(Tvector3_single,Tvector3_single):Tvector3_single

Synopsis: Add three-dimensional extended precision vectors together.

Declaration:
```
operator +(const x: Tvector3_single; const y: Tvector3_single)
                : Tvector3_single
```

Visibility: default

Description: This operator allows you to add two three-dimensional vectors with single precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.4.31 add(Tvector4_double,Double):Tvector4_double

Synopsis: Add scalar to four-dimensional double precision vector.

Declaration:
```
operator +(const x: Tvector4_double; y: Double) : Tvector4_double
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.4.32 add(Tvector4_double,Tvector4_double):Tvector4_double

Synopsis: Add four-dimensional double precision vectors together.

Declaration:
```
operator +(const x: Tvector4_double; const y: Tvector4_double)
                : Tvector4_double
```

Visibility: default

Description: This operator allows you to add two four-dimensional vectors with single precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.4.33 add(Tvector4_extended,extended):Tvector4_extended

Synopsis: Add scalar to four-dimensional extended precision vector.

Declaration:
```
operator +(const x: Tvector4_extended; y: extended) : Tvector4_extended
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.4.34 add(Tvector4_extended,Tvector4_extended):Tvector4_extended

Synopsis: Add four-dimensional extended precision vectors together.

Declaration:
```
operator +(const x: Tvector4_extended; const y: Tvector4_extended)
                : Tvector4_extended
```

Visibility: default

Description: This operator allows you to add two two-dimensional vectors with extended precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.4.35 add(Tvector4_single,single):Tvector4_single

Synopsis: Add scalar to four-dimensional single precision vector.

Declaration:
```
operator +(const x: Tvector4_single; y: single) : Tvector4_single
```

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.4.36 add(Tvector4_single,Tvector4_single):Tvector4_single

Synopsis: Add four-dimensional single precision vectors together.

Declaration:
```
operator +(const x: Tvector4_single; const y: Tvector4_single)
                : Tvector4_single
```

Visibility: default

Description: This operator allows you to add two four-dimensional vectors with single precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.4.37 assign(Tmatrix2_double):Tmatrix2_extended

Synopsis: Allow assignment of two-dimensional double precision matrix to two-dimensional extended precision matrix.

Declaration:
```
operator :=(const v: Tmatrix2_double) : Tmatrix2_extended
```

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a two-dimensional matrix with extended precision is expected.

64.4.38 assign(Tmatrix2_double):Tmatrix2_single

Synopsis: Allow assignment of two-dimensional double precision matrix to two-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix2_double) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a two-dimensional matrix with single precision is expected. Some accuracy is lost because of the conversion.

64.4.39 assign(Tmatrix2_double):Tmatrix3_double

Synopsis: Allow assignment of two-dimensional double precision matrix to three-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix2_double) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a three-dimensional matrix with double precision is expected. The extra fields are set to 0.

64.4.40 assign(Tmatrix2_double):Tmatrix3_extended

Synopsis: Allow assignment of two-dimensional double precision matrix to three-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix2_double) : Tmatrix3_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a three-dimensional matrix with extended precision is expected. The extra fields are set to 0.

64.4.41 assign(Tmatrix2_double):Tmatrix3_single

Synopsis: Allow assignment of two-dimensional single precision matrix to three-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix2_double) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a three-dimensional matrix with single precision is expected. The extra fields are set to 0 and some accuracy is lost because of the conversion.

64.4.42 assign(Tmatrix2_double):Tmatrix4_double

Synopsis: Allow assignment of two-dimensional double precision matrix to four-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix2_double) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a four-dimensional matrix with double precision is expected. The extra fields are set to 0.

64.4.43 assign(Tmatrix2_double):Tmatrix4_extended

Synopsis: Allow assignment of two-dimensional double precision matrix to four-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix2_double) : Tmatrix4_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a four-dimensional matrix with extended precision is expected. The extra fields are set to 0.

64.4.44 assign(Tmatrix2_double):Tmatrix4_single

Synopsis: Allow assignment of two-dimensional double precision matrix to four-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix2_double) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a four-dimensional matrix with single precision is expected. The extra fields are set to 0 and some precision is lost because of the conversion.

64.4.45 assign(Tmatrix2_extended):Tmatrix2_double

Synopsis: Allow assignment of two-dimensional extended precision matrix to two-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix2_extended) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to use a two-dimensional two with extended precision values wherever a two-dimensional matrix with double precision is expected. Some accuracy is lost because of the conversion.

64.4.46 assign(Tmatrix2_extended):Tmatrix2_single

Synopsis: Allow assignment of two-dimensional extended precision matrix to two-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix2_extended) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with extended precision values wherever a two-dimensional matrix with single precision is expected. Some accuracy is lost because of the conversion.

64.4.47 assign(Tmatrix2_extended):Tmatrix3_double

Synopsis: Allow assignment of two-dimensional extended precision matrix to three-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix2_extended) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with extended precision values wherever a three-dimensional matrix with double precision is expected. The extra fields are set to 0 and some accuracy is lost because of the conversion.

64.4.48 assign(Tmatrix2_extended):Tmatrix3_extended

Synopsis: Allow assignment of two-dimensional extended precision matrix to three-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix2_extended) : Tmatrix3_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with extended precision values wherever a three-dimensional matrix with extended precision is expected. The extra fields are set to 0.

64.4.49 assign(Tmatrix2_extended):Tmatrix3_single

Synopsis: Allow assignment of two-dimensional extended precision matrix to three-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix2_extended) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with extended precision values wherever a three-dimensional matrix with single precision is expected. The extra fields are set to 0 and some accuracy is lost because of the conversion.

64.4.50 assign(Tmatrix2_extended):Tmatrix4_double

Synopsis: Allow assignment of two-dimensional extended precision matrix to four-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix2_extended) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with extended precision values wherever a four-dimensional matrix with double precision is expected. The extra fields are set to 0 and some accuracy is lost because of the conversion.

64.4.51 assign(Tmatrix2_extended):Tmatrix4_extended

Synopsis: Allow assignment of two-dimensional extended precision matrix to four-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix2_extended) : Tmatrix4_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a four-dimensional matrix with single precision is expected. The extra fields are set to 0.

64.4.52 assign(Tmatrix2_extended):Tmatrix4_single

Synopsis: Allow assignment of two-dimensional extended precision matrix to four-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix2_extended) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with extended precision values wherever a four-dimensional matrix with single precision is expected. The extra fields are set to 0 and some precision is lost because of the conversion.

64.4.53 assign(Tmatrix2_single):Tmatrix2_double

Synopsis: Allow assignment of two-dimensional single precision matrix to two-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix2_single) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a two-dimensional matrix with double precision is expected.

64.4.54 assign(Tmatrix2_single):Tmatrix2_extended

Synopsis: Allow assignment of two-dimensional single precision matrix to two-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix2_single) : Tmatrix2_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a two-dimensional matrix with extended precision is expected.

64.4.55 assign(Tmatrix2_single):Tmatrix3_double

Synopsis: Allow assignment of two-dimensional single precision matrix to three-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix2_single) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a three-dimensional matrix with double precision is expected. The extra fields are set to 0.

64.4.56 assign(Tmatrix2_single):Tmatrix3_extended

Synopsis: Allow assignment of two-dimensional single precision matrix to three-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix2_single) : Tmatrix3_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a three-dimensional matrix with extended precision is expected. The extra fields are set to 0.

64.4.57 assign(Tmatrix2_single):Tmatrix3_single

Synopsis: Allow assignment of two-dimensional single precision matrix to three-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix2_single) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a three-dimensional matrix with single precision is expected. The extra fields are set to 0.

64.4.58 assign(Tmatrix2_single):Tmatrix4_double

Synopsis: Allow assignment of two-dimensional single precision matrix to four-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix2_single) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a four-dimensional matrix with double precision is expected. The extra fields are set to 0.

64.4.59 assign(Tmatrix2_single):Tmatrix4_extended

Synopsis: Allow assignment of two-dimensional single precision matrix to four-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix2_single) : Tmatrix4_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a four-dimensional matrix with extended precision is expected. The extra fields are set to 0.

64.4.60 assign(Tmatrix2_single):Tmatrix4_single

Synopsis: Allow assignment of two-dimensional single precision matrix to four-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix2_single) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a four-dimensional matrix with single precision is expected. The extra fields are set to 0.

64.4.61 assign(Tmatrix3_double):Tmatrix2_double

Synopsis: Allow assignment of three-dimensional double precision matrix to two-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix3_double) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a two-dimensional matrix with double precision is expected. The surplus fields are thrown away.

64.4.62 assign(Tmatrix3_double):Tmatrix2_extended

Synopsis: Allow assignment of three-dimensional double precision matrix to two-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix3_double) : Tmatrix2_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a two-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

64.4.63 assign(Tmatrix3_double):Tmatrix2_single

Synopsis: Allow assignment of three-dimensional double precision matrix to two-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix3_double) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away and some accuracy is lost because of the conversion.

64.4.64 assign(Tmatrix3_double):Tmatrix3_extended

Synopsis: Allow assignment of three-dimensional double precision matrix to three-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix3_double) : Tmatrix3_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a three-dimensional matrix with extended precision is expected.

64.4.65 assign(Tmatrix3_double):Tmatrix3_single

Synopsis: Allow assignment of three-dimensional double precision matrix to three-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix3_double) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a three-dimensional matrix with single precision is expected. Some precision is lost because of the conversion.

64.4.66 assign(Tmatrix3_double):Tmatrix4_double

Synopsis: Allow assignment of three-dimensional double precision matrix to four-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix3_double) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a four-dimensional matrix with double precision is expected.

64.4.67 assign(Tmatrix3_double):Tmatrix4_extended

Synopsis: Allow assignment of three-dimensional double precision matrix to four-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix3_double) : Tmatrix4_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a four-dimensional matrix with extended precision is expected.

64.4.68 assign(Tmatrix3_double):Tmatrix4_single

Synopsis: Allow assignment of three-dimensional double precision matrix to four-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix3_double) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a four-dimensional matrix with single precision is expected. Some precision is lost because of the conversion.

64.4.69 assign(Tmatrix3_extended):Tmatrix2_double

Synopsis: Allow assignment of three-dimensional extended precision matrix to two-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix3_extended) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a two-dimensional matrix with double precision is expected. The surplus fields are thrown away and some accuracy is lost because of the conversion.

64.4.70 assign(Tmatrix3_extended):Tmatrix2_extended

Synopsis: Allow assignment of three-dimensional extended precision matrix to two-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix3_extended) : Tmatrix2_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a two-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

64.4.71 assign(Tmatrix3_extended):Tmatrix2_single

Synopsis: Allow assignment of three-dimensional extended precision matrix to two-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix3_extended) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away and some precision is lost because of the conversion.

64.4.72 assign(Tmatrix3_extended):Tmatrix3_double

Synopsis: Allow assignment of three-dimensional extended precision matrix to three-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix3_extended) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a three-dimensional matrix with double precision is expected. Some precision is lost because of the conversion.

64.4.73 assign(Tmatrix3_extended):Tmatrix3_single

Synopsis: Allow assignment of three-dimensional extended precision matrix to three-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix3_extended) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a three-dimensional matrix with single precision is expected. Some precision is lost because of the conversion.

64.4.74 assign(Tmatrix3_extended):Tmatrix4_double

Synopsis: Allow assignment of three-dimensional extended precision matrix to four-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix3_extended) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a four-dimensional matrix with double precision is expected. Some precision is lost because of the conversion.

64.4.75 assign(Tmatrix3_extended):Tmatrix4_extended

Synopsis: Allow assignment of three-dimensional extended precision matrix to four-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix3_extended) : Tmatrix4_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a four-dimensional matrix with extended precision is expected.

64.4.76 assign(Tmatrix3_extended):Tmatrix4_single

Synopsis: Allow assignment of three-dimensional extended precision matrix to four-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix3_extended) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a four-dimensional matrix with single precision is expected. Some precision is lost because of the conversion.

64.4.77 assign(Tmatrix3_single):Tmatrix2_double

Synopsis: Allow assignment of three-dimensional single precision matrix to two-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix3_single) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a two-dimensional matrix with double precision is expected. The surplus fields are thrown away.

64.4.78 assign(Tmatrix3_single):Tmatrix2_extended

Synopsis: Allow assignment of three-dimensional single precision matrix to two-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix3_single) : Tmatrix2_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a two-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

64.4.79 assign(Tmatrix3_single):Tmatrix2_single

Synopsis: Allow assignment of three-dimensional single precision matrix to two-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix3_single) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away.

64.4.80 assign(Tmatrix3_single):Tmatrix3_double

Synopsis: Allow assignment of three-dimensional single precision matrix to three-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix3_single) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a three-dimensional matrix with double precision is expected.

64.4.81 assign(Tmatrix3_single):Tmatrix3_extended

Synopsis: Allow assignment of three-dimensional single precision matrix to three-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix3_single) : Tmatrix3_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a three-dimensional matrix with extended precision is expected.

64.4.82 assign(Tmatrix3_single):Tmatrix4_double

Synopsis: Allow assignment of three-dimensional single precision matrix to four-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix3_single) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a four-dimensional matrix with double precision is expected.

64.4.83 assign(Tmatrix3_single):Tmatrix4_extended

Synopsis: Allow assignment of three-dimensional single precision matrix to four-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix3_single) : Tmatrix4_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a four-dimensional matrix with extended precision is expected.

64.4.84 assign(Tmatrix3_single):Tmatrix4_single

Synopsis: Allow assignment of three-dimensional single precision matrix to four-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix3_single) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a four-dimensional matrix with single precision is expected.

64.4.85 assign(Tmatrix4_double):Tmatrix2_double

Synopsis: Allow assignment of four-dimensional double precision matrix to two-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix4_double) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a two-dimensional matrix with double precision is expected. The surplus fields are thrown away.

64.4.86 assign(Tmatrix4_double):Tmatrix2_extended

Synopsis: Allow assignment of four-dimensional double precision matrix to two-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix4_double) : Tmatrix2_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a two-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

64.4.87 assign(Tmatrix4_double):Tmatrix2_single

Synopsis: Allow assignment of four-dimensional double precision matrix to two-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix4_double) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away and some precision is lost in the conversion.

64.4.88 assign(Tmatrix4_double):Tmatrix3_double

Synopsis: Allow assignment of four-dimensional double precision matrix to three-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix4_double) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a three-dimensional matrix with double precision is expected. The surplus fields are thrown away.

64.4.89 assign(Tmatrix4_double):Tmatrix3_extended

Synopsis: Allow assignment of four-dimensional double precision matrix to three-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix4_double) : Tmatrix3_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a three-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

64.4.90 assign(Tmatrix4_double):Tmatrix3_single

Synopsis: Allow assignment of four-dimensional double precision matrix to three-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix4_double) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a three-dimensional matrix with single precision is expected. The surplus fields are thrown away and some precision is lost because of the conversion.

64.4.91 assign(Tmatrix4_double):Tmatrix4_extended

Synopsis: Allow assignment of four-dimensional double precision matrix to four-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix4_double) : Tmatrix4_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a four-dimensional matrix with extended precision is expected.

64.4.92 assign(Tmatrix4_double):Tmatrix4_single

Synopsis: Allow assignment of four-dimensional single precision matrix to four-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix4_double) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a four-dimensional matrix with single precision is expected. Some precision is lost because of the conversion.

64.4.93 assign(Tmatrix4_extended):Tmatrix2_double

Synopsis: Allow assignment of four-dimensional extended precision matrix to two-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix4_extended) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a two-dimensional matrix with double precision is expected. The surplus fields are thrown away and some precision is lost in the conversion.

64.4.94 assign(Tmatrix4_extended):Tmatrix2_extended

Synopsis: Allow assignment of four-dimensional extended precision matrix to two-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix4_extended) : Tmatrix2_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away.

64.4.95 assign(Tmatrix4_extended):Tmatrix2_single

Synopsis: Allow assignment of four-dimensional extended precision matrix to two-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix4_extended) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away and some precision is lost in the conversion.

64.4.96 assign(Tmatrix4_extended):Tmatrix3_double

Synopsis: Allow assignment of four-dimensional extended precision matrix to three-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix4_extended) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a three-dimensional matrix with double precision is expected. The surplus fields are thrown away.

64.4.97 assign(Tmatrix4_extended):Tmatrix3_extended

Synopsis: Allow assignment of four-dimensional extended precision matrix to three-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix4_extended) : Tmatrix3_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a three-dimensional matrix with double precision is expected. The surplus fields are thrown away.

64.4.98 assign(Tmatrix4_extended):Tmatrix3_single

Synopsis: Allow assignment of four-dimensional extended precision matrix to three-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix4_extended) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a three-dimensional matrix with single precision is expected. The surplus fields are thrown away and some precision is lost because of the conversion.

64.4.99 assign(Tmatrix4_extended):Tmatrix4_double

Synopsis: Allow assignment of four-dimensional extended precision matrix to four-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix4_extended) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a four-dimensional matrix with double precision is expected.

64.4.100 assign(Tmatrix4_extended):Tmatrix4_single

Synopsis: Allow assignment of four-dimensional extended precision matrix to four-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix4_extended) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a four-dimensional matrix with single precision is expected. Some precision is lost because of the conversion.

64.4.101 assign(Tmatrix4_single):Tmatrix2_double

Synopsis: Allow assignment of four-dimensional single precision matrix to two-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix4_single) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a two-dimensional matrix with double precision is expected. The surplus fields are thrown away.

64.4.102 assign(Tmatrix4_single):Tmatrix2_extended

Synopsis: Allow assignment of four-dimensional single precision matrix to two-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix4_single) : Tmatrix2_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a two-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

64.4.103 assign(Tmatrix4_single):Tmatrix2_single

Synopsis: Allow assignment of four-dimensional single precision matrix to two-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix4_single) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away.

64.4.104 assign(Tmatrix4_single):Tmatrix3_double

Synopsis: Allow assignment of four-dimensional single precision matrix to three-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix4_single) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a three-dimensional matrix with double precision is expected. The surplus fields are thrown away.

64.4.105 assign(Tmatrix4_single):Tmatrix3_extended

Synopsis: Allow assignment of four-dimensional single precision matrix to three-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix4_single) : Tmatrix3_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a three-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

64.4.106 assign(Tmatrix4_single):Tmatrix3_single

Synopsis: Allow assignment of four-dimensional single precision matrix to three-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix4_single) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a three-dimensional matrix with single precision is expected. The surplus fields are thrown away.

64.4.107 assign(Tmatrix4_single):Tmatrix4_double

Synopsis: Allow assignment of four-dimensional single precision matrix to four-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix4_single) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a four-dimensional matrix with double precision is expected.

64.4.108 assign(Tmatrix4_single):Tmatrix4_extended

Synopsis: Allow assignment of four-dimensional single precision matrix to four-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix4_single) : Tmatrix4_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a four-dimensional matrix with extended precision is expected.

64.4.109 assign(Tvector2_double):Tvector2_extended

Synopsis: Allow assignment of double precision vector to extended precision vector.

Declaration: `operator :=(const v: Tvector2_double) : Tvector2_extended`

Visibility: default

Description: This operator allows you to use a vector with double precision values wherever an extended precision vector is expected.

64.4.110 assign(Tvector2_double):Tvector2_single

Synopsis: Allow assignment of double precision vector to single precision vector.

Declaration: `operator :=(const v: Tvector2_double) : Tvector2_single`

Visibility: default

Description: This operator allows you to use a vector with double precision values wherever a single precision vector is expected, at the cost of loosing some precision.

64.4.111 assign(Tvector2_double):Tvector3_double

Synopsis: Allow assignment of two-dimensional double precision vector to three-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector2_double) : Tvector3_double`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with double precision values wherever a three-dimensional vector with double precision is expected. The third dimension is set to 0.0.

64.4.112 assign(Tvector2_double):Tvector3_extended

Synopsis: Allow assignment of two-dimensional double precision vector to three-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector2_double) : Tvector3_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with double precision values wherever a three-dimensional vector with extended precision is expected. The third dimension is set to 0.0.

64.4.113 assign(Tvector2_double):Tvector3_single

Synopsis: Allow assignment of two-dimensional double precision vector to three-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector2_double) : Tvector3_single`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with double precision values wherever a three-dimensional vector with single precision is expected. Some accuracy is lost because of the conversion and the third dimension is set to 0.0.

64.4.114 assign(Tvector2_double):Tvector4_double

Synopsis: Allow assignment of two-dimensional double precision vector to four-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector2_double) : Tvector4_double`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with double precision values wherever a four-dimensional vector with double precision is expected. The third and fourth dimensions are set to 0.0.

64.4.115 assign(Tvector2_double):Tvector4_extended

Synopsis: Allow assignment of two-dimensional double precision vector to four-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector2_double) : Tvector4_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a four-dimensional vector with extended precision is expected. The third and fourth dimensions are set to 0.0.

64.4.116 assign(Tvector2_double):Tvector4_single

Synopsis: Allow assignment of two-dimensional double precision vector to four-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector2_double) : Tvector4_single`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with double precision values wherever a four-dimensional vector with single precision is expected. Some accuracy is lost because of the conversion and the third and fourth dimensions are set to 0.0.

64.4.117 assign(Tvector2_extended):Tvector2_double

Synopsis: Allow assignment of extended precision vector to double precision vector.

Declaration: `operator :=(const v: Tvector2_extended) : Tvector2_double`

Visibility: default

Description: This operator allows you to use a vector with extended precision values wherever a double precision vector is expected, at the cost of loosing some precision.

64.4.118 assign(Tvector2_extended):Tvector2_single

Synopsis: Allow assignment of extended precision vector to single precision vector.

Declaration: `operator :=(const v: Tvector2_extended) : Tvector2_single`

Visibility: default

Description: This operator allows you to use a vector with extended precision values wherever a single precision vector is expected, at the cost of loosing some precision.

64.4.119 assign(Tvector2_extended):Tvector3_double

Synopsis: Allow assignment of two-dimensional extended precision vector to three-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector2_extended) : Tvector3_double`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with extended precision values wherever a three-dimensional vector with double precision is expected. Some accuracy is lost because of the conversion and the third dimension is set to 0.0.

64.4.120 assign(Tvector2_extended):Tvector3_extended

Synopsis: Allow assignment of two-dimensional extended precision vector to three-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector2_extended) : Tvector3_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with extended precision values wherever a three-dimensional vector with extended precision is expected. The third dimension is set to 0.0.

64.4.121 assign(Tvector2_extended):Tvector3_single

Synopsis: Allow assignment of two-dimensional extended precision vector to three-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector2_extended) : Tvector3_single`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with extended precision values wherever a three-dimensional vector with single precision is expected. Some accuracy is lost because of the conversion and the third dimension is set to 0.0.

64.4.122 assign(Tvector2_extended):Tvector4_double

Synopsis: Allow assignment of two-dimensional extended precision vector to four-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector2_extended) : Tvector4_double`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with extended precision values wherever a four-dimensional vector with double precision is expected. Some accuracy is lost because of the conversion and the third and fourth dimensions are set to 0.0.

64.4.123 assign(Tvector2_extended):Tvector4_extended

Synopsis: Allow assignment of two-dimensional extended precision vector to four-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector2_extended) : Tvector4_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with extended precision values wherever a four-dimensional vector with extended precision is expected. The third and fourth dimensions are set to 0.0.

64.4.124 assign(Tvector2_extended):Tvector4_single

Synopsis: Allow assignment of two-dimensional extended precision vector to four-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector2_extended) : Tvector4_single`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with extended precision values wherever a four-dimensional vector with single precision is expected. Some accuracy is lost because of the conversion and the third and fourth dimensions are set to 0.0.

64.4.125 assign(Tvector2_single):Tvector2_double

Synopsis: Allow assignment of single precision vector to double precision vector.

Declaration: `operator :=(const v: Tvector2_single) : Tvector2_double`

Visibility: default

Description: This operator allows you to use a vector with single precision values wherever a double precision vector is expected.

64.4.126 assign(Tvector2_single):Tvector2_extended

Synopsis: Allow assignment of single precision vector to extended precision vector.

Declaration: `operator :=(const v: Tvector2_single) : Tvector2_extended`

Visibility: default

Description: This operator allows you to use a vector with single precision values wherever an extended precision vector is expected.

64.4.127 assign(Tvector2_single):Tvector3_double

Synopsis: Allow assignment of two-dimensional single precision vector to three-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector2_single) : Tvector3_double`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a three-dimensional vector with double precision is expected. The third dimension is set to 0.0.

64.4.128 assign(Tvector2_single):Tvector3_extended

Synopsis: Allow assignment of two-dimensional single precision vector to three-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector2_single) : Tvector3_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a three-dimensional vector with extended precision is expected. The third dimension is set to 0.0.

64.4.129 assign(Tvector2_single):Tvector3_single

Synopsis: Allow assignment of two-dimensional single precision vector to three-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector2_single) : Tvector3_single`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a three-dimensional vector with single precision is expected. The third dimension is set to 0.0.

64.4.130 assign(Tvector2_single):Tvector4_double

Synopsis: Allow assignment of two-dimensional single precision vector to four-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector2_single) : Tvector4_double`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a four-dimensional vector with double precision is expected. The third and fourth dimensions are set to 0.0.

64.4.131 assign(Tvector2_single):Tvector4_extended

Synopsis: Allow assignment of two-dimensional single precision vector to four-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector2_single) : Tvector4_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a four-dimensional vector with extended precision is expected. The third and fourth dimensions are set to 0.0.

64.4.132 assign(Tvector2_single):Tvector4_single

Synopsis: Allow assignment of two-dimensional single precision vector to four-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector2_single) : Tvector4_single`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a four-dimensional vector with single precision is expected. The third and fourth dimensions are set to 0.0.

64.4.133 assign(Tvector3_double):Tvector2_double

Synopsis: Allow assignment of three-dimensional double precision vector to two-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector3_double) : Tvector2_double`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a two-dimensional vector with double precision is expected. The third dimension is thrown away.

64.4.134 assign(Tvector3_double):Tvector2_extended

Synopsis: Allow assignment of three-dimensional double precision vector to two-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector3_double) : Tvector2_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a two-dimensional vector with extended precision is expected. The third dimension is thrown away.

64.4.135 assign(Tvector3_double):Tvector2_single

Synopsis: Allow assignment of three-dimensional double precision vector to two-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector3_double) : Tvector2_single`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a two-dimensional vector with single precision is expected. The third dimension is thrown away and some precision is lost because of the conversion.

64.4.136 assign(Tvector3_double):Tvector3_extended

Synopsis: Allow assignment of three-dimensional double precision vector to three-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector3_double) : Tvector3_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a three-dimensional vector with extended precision is expected.

64.4.137 assign(Tvector3_double):Tvector3_single

Synopsis: Allow assignment of three-dimensional double precision vector to three-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector3_double) : Tvector3_single`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a three-dimensional vector with single precision is expected. Some precision is lost because of the conversion.

64.4.138 assign(Tvector3_double):Tvector4_double

Synopsis: Allow assignment of three-dimensional double precision vector to four-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector3_double) : Tvector4_double`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a four-dimensional vector with double precision is expected. The fourth dimension is set to 0.

64.4.139 assign(Tvector3_double):Tvector4_extended

Synopsis: Allow assignment of three-dimensional double precision vector to four-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector3_double) : Tvector4_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a four-dimensional vector with extended precision is expected. The fourth dimension is set to 0.

64.4.140 assign(Tvector3_double):Tvector4_single

Synopsis: Allow assignment of three-dimensional double precision vector to four-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector3_double) : Tvector4_single`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a four-dimensional vector with double precision is expected. The fourth dimension is set to 0 and some precision is lost because of the conversion.

64.4.141 assign(Tvector3_extended):Tvector2_double

Synopsis: Allow assignment of three-dimensional extended precision vector to two-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector3_extended) : Tvector2_double`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a two-dimensional vector with double precision is expected. The third dimension is thrown away and some precision is lost because of the conversion.

64.4.142 assign(Tvector3_extended):Tvector2_extended

Synopsis: Allow assignment of three-dimensional extended precision vector to two-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector3_extended) : Tvector2_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a two-dimensional vector with extended precision is expected. The third dimension is thrown away.

64.4.143 assign(Tvector3_extended):Tvector2_single

Synopsis: Allow assignment of three-dimensional extended precision vector to two-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector3_extended) : Tvector2_single`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a two-dimensional vector with single precision is expected. The third dimension is thrown away and some precision is lost because of the conversion.

64.4.144 assign(Tvector3_extended):Tvector3_double

Synopsis: Allow assignment of three-dimensional extended precision vector to three-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector3_extended) : Tvector3_double`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a three-dimensional vector with double precision is expected. Some precision is lost because of the conversion.

64.4.145 assign(Tvector3_extended):Tvector3_single

Synopsis: Allow assignment of three-dimensional single precision vector to three-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector3_extended) : Tvector3_single`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a three-dimensional vector with double precision is expected. Some precision is lost because of the conversion.

64.4.146 assign(Tvector3_extended):Tvector4_double

Synopsis: Allow assignment of three-dimensional extended precision vector to four-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector3_extended) : Tvector4_double`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a four-dimensional vector with double precision is expected. The fourth dimension is set to 0 and some accuracy is lost because of the conversion.

64.4.147 assign(Tvector3_extended):Tvector4_extended

Synopsis: Allow assignment of three-dimensional extended precision vector to four-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector3_extended) : Tvector4_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a four-dimensional vector with extended precision is expected. The fourth dimension is set to 0.

64.4.148 assign(Tvector3_extended):Tvector4_single

Synopsis: Allow assignment of three-dimensional extended precision vector to four-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector3_extended) : Tvector4_single`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a four-dimensional vector with single precision is expected. The fourth dimension is set to 0 and some accuracy is lost because of the conversion.

64.4.149 assign(Tvector3_single):Tvector2_double

Synopsis: Allow assignment of three-dimensional single precision vector to two-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector3_single) : Tvector2_double`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a two-dimensional vector with double precision is expected. The third dimension is thrown away.

64.4.150 assign(Tvector3_single):Tvector2_extended

Synopsis: Allow assignment of three-dimensional single precision vector to two-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector3_single) : Tvector2_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a two-dimensional vector with extended precision is expected. The third dimension is thrown away.

64.4.151 assign(Tvector3_single):Tvector2_single

Synopsis: Allow assignment of three-dimensional single precision vector to two-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector3_single) : Tvector2_single`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a two-dimensional vector with single precision is expected. The third dimension is thrown away.

64.4.152 assign(Tvector3_single):Tvector3_double

Synopsis: Allow assignment of three-dimensional single precision vector to three-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector3_single) : Tvector3_double`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a three-dimensional vector with double precision is expected.

64.4.153 assign(Tvector3_single):Tvector3_extended

Synopsis: Allow assignment of three-dimensional single precision vector to three-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector3_single) : Tvector3_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a three-dimensional vector with extended precision is expected.

64.4.154 assign(Tvector3_single):Tvector4_double

Synopsis: Allow assignment of three-dimensional single precision vector to four-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector3_single) : Tvector4_double`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a four-dimensional vector with double precision is expected. The fourth dimension is set to 0.

64.4.155 assign(Tvector3_single):Tvector4_extended

Synopsis: Allow assignment of three-dimensional single precision vector to four-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector3_single) : Tvector4_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a four-dimensional vector with extended precision is expected. The fourth dimension is set to 0.

64.4.156 assign(Tvector3_single):Tvector4_single

Synopsis: Allow assignment of three-dimensional single precision vector to four-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector3_single) : Tvector4_single`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a four-dimensional vector with single precision is expected. The fourth dimension is set to 0.

64.4.157 assign(Tvector4_double):Tvector2_double

Synopsis: Allow assignment of four-dimensional double precision vector to two-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector4_double) : Tvector2_double`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a two-dimensional vector with double precision is expected. The third and fourth dimensions are thrown away.

64.4.158 assign(Tvector4_double):Tvector2_extended

Synopsis: Allow assignment of four-dimensional double precision vector to two-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector4_double) : Tvector2_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a two-dimensional vector with extended precision is expected. The third and fourth dimensions are thrown away.

64.4.159 assign(Tvector4_double):Tvector2_single

Synopsis: Allow assignment of four-dimensional double precision vector to two-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector4_double) : Tvector2_single`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a two-dimensional vector with single precision is expected. The third and fourth dimensions are thrown away and some accuracy is lost because of the conversion.

64.4.160 assign(Tvector4_double):Tvector3_double

Synopsis: Allow assignment of four-dimensional double precision vector to three-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector4_double) : Tvector3_double`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a three-dimensional vector with double precision is expected. The fourth dimension is thrown away.

64.4.161 assign(Tvector4_double):Tvector3_extended

Synopsis: Allow assignment of four-dimensional double precision vector to three-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector4_double) : Tvector3_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a three-dimensional vector with extended precision is expected. The fourth dimension is thrown away.

64.4.162 assign(Tvector4_double):Tvector3_single

Synopsis: Allow assignment of four-dimensional double precision vector to three-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector4_double) : Tvector3_single`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a three-dimensional vector with single precision is expected. The fourth dimension is thrown away and some accuracy is lost because of the conversion.

64.4.163 assign(Tvector4_double):Tvector4_extended

Synopsis: Allow assignment of four-dimensional single precision vector to four-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector4_double) : Tvector4_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a four-dimensional vector with extended precision is expected.

64.4.164 assign(Tvector4_double):Tvector4_single

Synopsis: Allow assignment of four-dimensional double precision vector to four-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector4_double) : Tvector4_single`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a four-dimensional vector with single precision is expected. Some accuracy is lost because of the conversion.

64.4.165 assign(Tvector4_extended):Tvector2_double

Synopsis: Allow assignment of four-dimensional extended precision vector to two-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector4_extended) : Tvector2_double`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a two-dimensional vector with double precision is expected. The third and fourth dimensions are thrown away and some accuracy is lost because of the conversion.

64.4.166 assign(Tvector4_extended):Tvector2_extended

Synopsis: Allow assignment of four-dimensional extended precision vector to two-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector4_extended) : Tvector2_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a two-dimensional vector with extended precision is expected. The third and fourth dimensions are thrown away.

64.4.167 assign(Tvector4_extended):Tvector2_single

Synopsis: Allow assignment of four-dimensional extended precision vector to two-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector4_extended) : Tvector2_single`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a two-dimensional vector with single precision is expected. The third and fourth dimensions are thrown away and some accuracy is lost because of the conversion.

64.4.168 assign(Tvector4_extended):Tvector3_double

Synopsis: Allow assignment of four-dimensional extended precision vector to three-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector4_extended) : Tvector3_double`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a three-dimensional vector with double precision is expected. The fourth dimension is thrown away and some accuracy is lost because of the conversion.

64.4.169 assign(Tvector4_extended):Tvector3_extended

Synopsis: Allow assignment of four-dimensional extended precision vector to three-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector4_extended) : Tvector3_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a three-dimensional vector with extended precision is expected. The fourth dimensions are thrown away.

64.4.170 assign(Tvector4_extended):Tvector3_single

Synopsis: Allow assignment of four-dimensional extended precision vector to three-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector4_extended) : Tvector3_single`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a three-dimensional vector with single precision is expected. The fourth dimension is thrown away and some accuracy is lost because of the conversion.

64.4.171 assign(Tvector4_extended):Tvector4_double

Synopsis: Allow assignment of four-dimensional single precision vector to four-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector4_extended) : Tvector4_double`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a four-dimensional vector with double precision is expected. Some accuracy is lost because of the conversion.

64.4.172 assign(Tvector4_extended):Tvector4_single

Synopsis: Allow assignment of four-dimensional extended precision vector to four-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector4_extended) : Tvector4_single`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a four-dimensional vector with single precision is expected. Some accuracy is lost because of the conversion.

64.4.173 assign(Tvector4_single):Tvector2_double

Synopsis: Allow assignment of four-dimensional single precision vector to two-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector4_single) : Tvector2_double`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a two-dimensional vector with double precision is expected. The third and fourth dimensions are thrown away.

64.4.174 assign(Tvector4_single):Tvector2_extended

Synopsis: Allow assignment of four-dimensional single precision vector to two-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector4_single) : Tvector2_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a two-dimensional vector with extended precision is expected. The third and fourth dimensions are thrown away.

64.4.175 assign(Tvector4_single):Tvector2_single

Synopsis: Allow assignment of four-dimensional single precision vector to two-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector4_single) : Tvector2_single`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a two-dimensional vector with single precision is expected. The third and fourth dimensions are thrown away.

64.4.176 assign(Tvector4_single):Tvector3_double

Synopsis: Allow assignment of four-dimensional single precision vector to three-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector4_single) : Tvector3_double`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a three-dimensional vector with double precision is expected. The fourth dimension is thrown away.

64.4.177 assign(Tvector4_single):Tvector3_extended

Synopsis: Allow assignment of four-dimensional single precision vector to three-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector4_single) : Tvector3_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a three-dimensional vector with extended precision is expected. The fourth dimension is thrown away.

64.4.178 assign(Tvector4_single):Tvector3_single

Synopsis: Allow assignment of four-dimensional single precision vector to three-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector4_single) : Tvector3_single`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a three-dimensional vector with single precision is expected. The fourth dimension is thrown away.

64.4.179 assign(Tvector4_single):Tvector4_double

Synopsis: Allow assignment of four-dimensional single precision vector to four-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector4_single) : Tvector4_double`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a four-dimensional vector with double precision is expected.

64.4.180 assign(Tvector4_single):Tvector4_extended

Synopsis: Allow assignment of four-dimensional single precision vector to four-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector4_single) : Tvector4_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a four-dimensional vector with extended precision is expected.

64.4.181 divide(Tmatrix2_double,Double):Tmatrix2_double

Synopsis: Divide a two-dimensional single precision matrix by a scalar.

Declaration:
```
operator /(const m: Tmatrix2_double; const x: Double) : Tmatrix2_double
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.4.182 divide(Tmatrix2_extended,extended):Tmatrix2_extended

Synopsis: Divide a two-dimensional single precision matrix by a scalar.

Declaration:
```
operator /(const m: Tmatrix2_extended; const x: extended)
                : Tmatrix2_extended
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.4.183 divide(Tmatrix2_single,single):Tmatrix2_single

Synopsis: Divide a two-dimensional single precision matrix by a scalar.

Declaration:
```
operator /(const m: Tmatrix2_single; const x: single) : Tmatrix2_single
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.4.184 divide(Tmatrix3_double,Double):Tmatrix3_double

Synopsis: Divide a two-dimensional single precision matrix by a scalar.

Declaration:
```
operator /(const m: Tmatrix3_double; const x: Double) : Tmatrix3_double
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.4.185 divide(Tmatrix3_extended,extended):Tmatrix3_extended

Synopsis: Divide a two-dimensional single precision matrix by a scalar.

Declaration:
```
operator /(const m: Tmatrix3_extended; const x: extended)
                : Tmatrix3_extended
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.4.186 divide(Tmatrix3_single,single):Tmatrix3_single

Synopsis: Divide a two-dimensional single precision matrix by a scalar.

Declaration:
```
operator /(const m: Tmatrix3_single; const x: single) : Tmatrix3_single
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.4.187 divide(Tmatrix4_double,Double):Tmatrix4_double

Synopsis: Divide a two-dimensional single precision matrix by a scalar.

Declaration:
```
operator /(const m: Tmatrix4_double; const x: Double) : Tmatrix4_double
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.4.188 divide(Tmatrix4_extended,extended):Tmatrix4_extended

Synopsis: Divide a two-dimensional single precision matrix by a scalar.

Declaration:
```
operator /(const m: Tmatrix4_extended; const x: extended)
                : Tmatrix4_extended
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.4.189 divide(Tmatrix4_single,single):Tmatrix4_single

Synopsis: Divide a two-dimensional single precision matrix by a scalar.

Declaration:
```
operator /(const m: Tmatrix4_single; const x: single) : Tmatrix4_single
```

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.4.190 divide(Tvector2_double,Double):Tvector2_double

Synopsis: Divide a two-dimensional double precision vector by a scalar.

Declaration:
```
operator /(const x: Tvector2_double; y: Double) : Tvector2_double
```

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.4.191 divide(Tvector2_extended,extended):Tvector2_extended

Synopsis: Divide a two-dimensional extended precision vector by a scalar.

Declaration: `operator /(const x: Tvector2_extended; y: extended) : Tvector2_extended`

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.4.192 divide(Tvector2_single,single):Tvector2_single

Synopsis: Divide a two-dimensional single precision vector by a scalar.

Declaration: `operator /(const x: Tvector2_single; y: single) : Tvector2_single`

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.4.193 divide(Tvector3_double,Double):Tvector3_double

Synopsis: Divide a three-dimensional double precision vector by a scalar.

Declaration: `operator /(const x: Tvector3_double; y: Double) : Tvector3_double`

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.4.194 divide(Tvector3_extended,extended):Tvector3_extended

Synopsis: Divide a three-dimensional extended precision vector by a scalar.

Declaration: `operator /(const x: Tvector3_extended; y: extended) : Tvector3_extended`

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.4.195 divide(Tvector3_single,single):Tvector3_single

Synopsis: Divide a three-dimensional single precision vector by a scalar.

Declaration: `operator /(const x: Tvector3_single; y: single) : Tvector3_single`

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.4.196 divide(Tvector4_double,Double):Tvector4_double

Synopsis: Divide a four-dimensional double precision vector by a scalar.

Declaration:
```
operator /(const x: Tvector4_double; y: Double) : Tvector4_double
```

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.4.197 divide(Tvector4_extended,extended):Tvector4_extended

Synopsis: Divide a four-dimensional extended precision vector by a scalar.

Declaration:
```
operator /(const x: Tvector4_extended; y: extended) : Tvector4_extended
```

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.4.198 divide(Tvector4_single,single):Tvector4_single

Synopsis: Divide a four-dimensional single precision vector by a scalar.

Declaration:
```
operator /(const x: Tvector4_single; y: single) : Tvector4_single
```

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.4.199 multiply(Tmatrix2_double,Double):Tmatrix2_double

Synopsis: Multiply a two-dimensional double precision matrix by a scalar.

Declaration:
```
operator *(const m: Tmatrix2_double; const x: Double) : Tmatrix2_double
```

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.4.200 multiply(Tmatrix2_double,Tmatrix2_double):Tmatrix2_double

Synopsis: Give product of two two-dimensional double precision matrices.

Declaration:
```
operator *(const m1: Tmatrix2_double; const m2: Tmatrix2_double)
                : Tmatrix2_double
```

Visibility: default

Description: This operator allows you to multiply two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.4.201 multiply(Tmatrix2_double,Tvector2_double):Tvector2_double

Synopsis: Give product of a two-dimensional double precision matrix and vector.

Declaration:
```
operator *(const m: Tmatrix2_double; const v: Tvector2_double)
  : Tvector2_double
```

Visibility: default

Description: This operator allows you to multiply a two-dimensional double precision matrices with a two dimensional double precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.4.202 multiply(Tmatrix2_extended,extended):Tmatrix2_extended

Synopsis: Multiply a two-dimensional extended precision matrix by a scalar.

Declaration:
```
operator *(const m: Tmatrix2_extended; const x: extended)
  : Tmatrix2_extended
```

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.4.203 multiply(Tmatrix2_extended,Tmatrix2_extended):Tmatrix2_extended

Synopsis: Give product of two two-dimensional extended precision matrices.

Declaration:
```
operator *(const m1: Tmatrix2_extended; const m2: Tmatrix2_extended)
  : Tmatrix2_extended
```

Visibility: default

Description: This operator allows you to multiply two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.4.204 multiply(Tmatrix2_extended,Tvector2_extended):Tvector2_extended

Synopsis: Give product of a two-dimensional extended precision matrix and vector.

Declaration:
```
operator *(const m: Tmatrix2_extended; const v: Tvector2_extended)
  : Tvector2_extended
```

Visibility: default

Description: This operator allows you to multiply a two-dimensional extended precision matrices with a two dimensional extended precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.4.205 multiply(Tmatrix2_single,single):Tmatrix2_single

Synopsis: Multiply a two-dimensional single precision matrix by a scalar.

Declaration:
```
operator *(const m: Tmatrix2_single; const x: single) : Tmatrix2_single
```

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.4.206 multiply(Tmatrix2_single,Tmatrix2_single):Tmatrix2_single

Synopsis: Give product of two two-dimensional single precision matrices.

Declaration:
```
operator *(const m1: Tmatrix2_single; const m2: Tmatrix2_single)
  : Tmatrix2_single
```

Visibility: default

Description: This operator allows you to multiply two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.4.207 multiply(Tmatrix2_single,Tvector2_single):Tvector2_single

Synopsis: Give product of a two-dimensional single precision matrix and vector.

Declaration:
```
operator *(const m: Tmatrix2_single; const v: Tvector2_single)
  : Tvector2_single
```

Visibility: default

Description: This operator allows you to multiply a two-dimensional single precision matrices with a two dimensional single precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.4.208 multiply(Tmatrix3_double,Double):Tmatrix3_double

Synopsis: Multiply a three-dimensional double precision matrix by a scalar.

Declaration:
```
operator *(const m: Tmatrix3_double; const x: Double) : Tmatrix3_double
```

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.4.209 multiply(Tmatrix3_double,Tmatrix3_double):Tmatrix3_double

Synopsis: Give product of two three-dimensional double precision matrices.

Declaration:
```
operator *(const m1: Tmatrix3_double; const m2: Tmatrix3_double)
  : Tmatrix3_double
```

Visibility: default

Description: This operator allows you to multiply two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.4.210 multiply(Tmatrix3_double,Tvector3_double):Tvector3_double

Synopsis: Give product of a three-dimensional double precision matrix and vector.

Declaration:
```
operator *(const m: Tmatrix3_double; const v: Tvector3_double)
  : Tvector3_double
```

Visibility: default

Description: This operator allows you to multiply a three-dimensional double precision matrices with a three dimensional double precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.4.211 multiply(Tmatrix3_extended,extended):Tmatrix3_extended

Synopsis: Multiply a three-dimensional extended precision matrix by a scalar.

Declaration:
```
operator *(const m: Tmatrix3_extended; const x: extended)
  : Tmatrix3_extended
```

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.4.212 multiply(Tmatrix3_extended,Tmatrix3_extended):Tmatrix3_extended

Synopsis: Give product of two three-dimensional extended precision matrices.

Declaration:
```
operator *(const m1: Tmatrix3_extended; const m2: Tmatrix3_extended)
  : Tmatrix3_extended
```

Visibility: default

Description: This operator allows you to multiply two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.4.213 multiply(Tmatrix3_extended,Tvector3_extended):Tvector3_extended

Synopsis: Give product of a three-dimensional extended precision matrix and vector.

Declaration:
```
operator *(const m: Tmatrix3_extended; const v: Tvector3_extended)
  : Tvector3_extended
```

Visibility: default

Description: This operator allows you to multiply a three-dimensional extended precision matrices with a three dimensional extended precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.4.214 multiply(Tmatrix3_single,single):Tmatrix3_single

Synopsis: Multiply a three-dimensional single precision matrix by a scalar.

Declaration:
```
operator *(const m: Tmatrix3_single; const x: single) : Tmatrix3_single
```

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.4.215 multiply(Tmatrix3_single,Tmatrix3_single):Tmatrix3_single

Synopsis: Give product of two three-dimensional single precision matrices.

Declaration:
```
operator *(const m1: Tmatrix3_single; const m2: Tmatrix3_single)
  : Tmatrix3_single
```

Visibility: default

Description: This operator allows you to multiply two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.4.216 multiply(Tmatrix3_single,Tvector3_single):Tvector3_single

Synopsis: Give product of a three-dimensional single precision matrix and vector.

Declaration:
```
operator *(const m: Tmatrix3_single; const v: Tvector3_single)
  : Tvector3_single
```

Visibility: default

Description: This operator allows you to multiply a three-dimensional single precision matrices with a three dimensional single precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.4.217 multiply(Tmatrix4_double,Double):Tmatrix4_double

Synopsis: Multiply a four-dimensional double precision matrix by a scalar.

Declaration:
```
operator *(const m: Tmatrix4_double; const x: Double) : Tmatrix4_double
```

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.4.218 multiply(Tmatrix4_double,Tmatrix4_double):Tmatrix4_double

Synopsis: Give product of two four-dimensional double precision matrices.

Declaration:
```
operator *(const m1: Tmatrix4_double; const m2: Tmatrix4_double)
  : Tmatrix4_double
```

Visibility: default

Description: This operator allows you to multiply two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.4.219 multiply(Tmatrix4_double,Tvector4_double):Tvector4_double

Synopsis: Give product of a four-dimensional double precision matrix and vector.

Declaration:
```
operator *(const m: Tmatrix4_double; const v: Tvector4_double)
  : Tvector4_double
```

Visibility: default

Description: This operator allows you to multiply a four-dimensional double precision matrices with a four dimensional double precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.4.220 multiply(Tmatrix4_extended,extended):Tmatrix4_extended

Synopsis: Multiply a four-dimensional extended precision matrix by a scalar.

Declaration:
```
operator *(const m: Tmatrix4_extended; const x: extended)
  : Tmatrix4_extended
```

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.4.221 multiply(Tmatrix4_extended,Tmatrix4_extended):Tmatrix4_extended

Synopsis: Give product of two four-dimensional extended precision matrices.

Declaration:
```
operator *(const m1: Tmatrix4_extended; const m2: Tmatrix4_extended)
  : Tmatrix4_extended
```

Visibility: default

Description: This operator allows you to multiply two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.4.222 multiply(Tmatrix4_extended,Tvector4_extended):Tvector4_extended

Synopsis: Give product of a four-dimensional extended precision matrix and vector.

Declaration:
```
operator *(const m: Tmatrix4_extended; const v: Tvector4_extended)
  : Tvector4_extended
```

Visibility: default

Description: This operator allows you to multiply a four-dimensional extended precision matrices with a four dimensional extended precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.4.223 multiply(Tmatrix4_single,single):Tmatrix4_single

Synopsis: Multiply a four-dimensional single precision matrix by a scalar.

Declaration:
```
operator *(const m: Tmatrix4_single; const x: single) : Tmatrix4_single
```

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.4.224 multiply(Tmatrix4_single,Tmatrix4_single):Tmatrix4_single

Synopsis: Give product of two four-dimensional single precision matrices.

Declaration:
```
operator *(const m1: Tmatrix4_single; const m2: Tmatrix4_single)
                  : Tmatrix4_single
```

Visibility: default

Description: This operator allows you to multiply two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.4.225 multiply(Tmatrix4_single,Tvector4_single):Tvector4_single

Synopsis: Give product of a four-dimensional single precision matrix and vector.

Declaration:
```
operator *(const m: Tmatrix4_single; const v: Tvector4_single)
                  : Tvector4_single
```

Visibility: default

Description: This operator allows you to multiply a four-dimensional single precision matrices with a four dimensional single precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.4.226 multiply(Tvector2_double,Double):Tvector2_double

Synopsis: Multiply a two-dimensional double precision vector by a scalar.

Declaration:
```
operator *(const x: Tvector2_double; y: Double) : Tvector2_double
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.4.227 multiply(Tvector2_double,Tvector2_double):Tvector2_double

Synopsis: Multiply two vectors element wise.

Declaration:
```
operator *(const x: Tvector2_double; const y: Tvector2_double)
                  : Tvector2_double
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.4.228 multiply(Tvector2_extended,extended):Tvector2_extended

Synopsis: Multiply a two-dimensional extended precision vector by a scalar.

Declaration:
```
operator *(const x: Tvector2_extended; y: extended) : Tvector2_extended
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.4.229 multiply(Tvector2_extended,Tvector2_extended):Tvector2_extended

Synopsis: Multiply two vectors element wise.

Declaration:
```
operator *(const x: Tvector2_extended; const y: Tvector2_extended)
  : Tvector2_extended
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.4.230 multiply(Tvector2_single,single):Tvector2_single

Synopsis: Multiply a two-dimensional single precision vector by a scalar.

Declaration:
```
operator *(const x: Tvector2_single; y: single) : Tvector2_single
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.4.231 multiply(Tvector2_single,Tvector2_single):Tvector2_single

Synopsis: Multiply two vectors element wise.

Declaration:
```
operator *(const x: Tvector2_single; const y: Tvector2_single)
  : Tvector2_single
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.4.232 multiply(Tvector3_double,Double):Tvector3_double

Synopsis: Multiply a three-dimensional double precision vector by a scalar.

Declaration:
```
operator *(const x: Tvector3_double; y: Double) : Tvector3_double
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.4.233 multiply(Tvector3_double,Tvector3_double):Tvector3_double

Synopsis: Multiply two vectors element wise.

Declaration:
```
operator *(const x: Tvector3_double; const y: Tvector3_double)
  : Tvector3_double
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.4.234 multiply(Tvector3_extended,extended):Tvector3_extended

Synopsis: Multiply a three-dimensional extended precision vector by a scalar.

Declaration:
```
operator *(const x: Tvector3_extended; y: extended) : Tvector3_extended
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.4.235 multiply(Tvector3_extended,Tvector3_extended):Tvector3_extended

Synopsis: Multiply two vectors element wise.

Declaration:
```
operator *(const x: Tvector3_extended; const y: Tvector3_extended)
  : Tvector3_extended
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.4.236 multiply(Tvector3_single,single):Tvector3_single

Synopsis: Multiply a three-dimensional single precision vector by a scalar.

Declaration:
```
operator *(const x: Tvector3_single; y: single) : Tvector3_single
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.4.237 multiply(Tvector3_single,Tvector3_single):Tvector3_single

Synopsis: Multiply two vectors element wise.

Declaration:
```
operator *(const x: Tvector3_single; const y: Tvector3_single)
  : Tvector3_single
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.4.238 multiply(Tvector4_double,Double):Tvector4_double

Synopsis: Multiply a four-dimensional double precision vector by a scalar.

Declaration:
```
operator *(const x: Tvector4_double; y: Double) : Tvector4_double
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.4.239 multiply(Tvector4_double,Tvector4_double):Tvector4_double

Synopsis: Multiply two vectors element wise.

Declaration:
```
operator *(const x: Tvector4_double; const y: Tvector4_double)
  : Tvector4_double
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.4.240 multiply(Tvector4_extended,extended):Tvector4_extended

Synopsis: Multiply a four-dimensional extended precision vector by a scalar.

Declaration:
```
operator *(const x: Tvector4_extended; y: extended) : Tvector4_extended
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.4.241 multiply(Tvector4_extended,Tvector4_extended):Tvector4_extended

Synopsis: Multiply two vectors element wise.

Declaration:
```
operator *(const x: Tvector4_extended; const y: Tvector4_extended)
  : Tvector4_extended
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.4.242 multiply(Tvector4_single,single):Tvector4_single

Synopsis: Multiply a four-dimensional single precision vector by a scalar.

Declaration:
```
operator *(const x: Tvector4_single; y: single) : Tvector4_single
```

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.4.243 multiply(Tvector4_single,Tvector4_single):Tvector4_single

Synopsis: Multiply two vectors element wise.

Declaration:
```
operator *(const x: Tvector4_single; const y: Tvector4_single)
                 : Tvector4_single
```

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.4.244 negative(Tmatrix2_double):Tmatrix2_double

Synopsis: Negate two-dimensional double precision matrix.

Declaration:
```
operator -(const m1: Tmatrix2_double) : Tmatrix2_double
```

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.4.245 negative(Tmatrix2_extended):Tmatrix2_extended

Synopsis: Negate two-dimensional extended precision matrix.

Declaration:
```
operator -(const m1: Tmatrix2_extended) : Tmatrix2_extended
```

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.4.246 negative(Tmatrix2_single):Tmatrix2_single

Synopsis: Negate two-dimensional single precision matrix.

Declaration:
```
operator -(const m1: Tmatrix2_single) : Tmatrix2_single
```

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.4.247 negative(Tmatrix3_double):Tmatrix3_double

Synopsis: Negate three-dimensional double precision matrix.

Declaration:
```
operator -(const m1: Tmatrix3_double) : Tmatrix3_double
```

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.4.248 negative(Tmatrix3_extended):Tmatrix3_extended

Synopsis: Negate three-dimensional extended precision matrix.

Declaration:
```
operator -(const m1: Tmatrix3_extended) : Tmatrix3_extended
```

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.4.249 negative(Tmatrix3_single):Tmatrix3_single

Synopsis: Negate three-dimensional single precision matrix.

Declaration: `operator -(const m1: Tmatrix3_single) : Tmatrix3_single`

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.4.250 negative(Tmatrix4_double):Tmatrix4_double

Synopsis: Negate four-dimensional double precision matrix.

Declaration: `operator -(const m1: Tmatrix4_double) : Tmatrix4_double`

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.4.251 negative(Tmatrix4_extended):Tmatrix4_extended

Synopsis: Negate four-dimensional extended precision matrix.

Declaration: `operator -(const m1: Tmatrix4_extended) : Tmatrix4_extended`

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.4.252 negative(Tmatrix4_single):Tmatrix4_single

Synopsis: Negate four-dimensional single precision matrix.

Declaration: `operator -(const m1: Tmatrix4_single) : Tmatrix4_single`

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.4.253 negative(Tvector2_double):Tvector2_double

Synopsis: Negate two-dimensional vector.

Declaration: `operator -(const x: Tvector2_double) : Tvector2_double`

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.4.254 negative(Tvector2_extended):Tvector2_extended

Synopsis: Negate two-dimensional vector.

Declaration: `operator -(const x: Tvector2_extended) : Tvector2_extended`

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.4.255 negative(Tvector2_single):Tvector2_single

Synopsis: Negate two-dimensional vector.

Declaration: `operator -(const x: Tvector2_single) : Tvector2_single`

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.4.256 negative(Tvector3_double):Tvector3_double

Synopsis: Negate three-dimensional vector.

Declaration: `operator -(const x: Tvector3_double) : Tvector3_double`

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.4.257 negative(Tvector3_extended):Tvector3_extended

Synopsis: Negate three-dimensional vector.

Declaration: `operator -(const x: Tvector3_extended) : Tvector3_extended`

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.4.258 negative(Tvector3_single):Tvector3_single

Synopsis: Negate three-dimensional vector.

Declaration: `operator -(const x: Tvector3_single) : Tvector3_single`

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.4.259 negative(Tvector4_double):Tvector4_double

Synopsis: Negate four-dimensional vector.

Declaration: `operator -(const x: Tvector4_double) : Tvector4_double`

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.4.260 negative(Tvector4_extended):Tvector4_extended

Synopsis: Negate four-dimensional vector.

Declaration:
```
operator -(const x: Tvector4_extended) : Tvector4_extended
```

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.4.261 negative(Tvector4_single):Tvector4_single

Synopsis: Negate four-dimensional vector.

Declaration:
```
operator -(const x: Tvector4_single) : Tvector4_single
```

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.4.262 power(Tvector2_double,Tvector2_double):Double

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
operator **(const x: Tvector2_double; const y: Tvector2_double) : Double
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.4.263 power(Tvector2_extended,Tvector2_extended):extended

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
operator **(const x: Tvector2_extended; const y: Tvector2_extended)
                  : extended
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.4.264 power(Tvector2_single,Tvector2_single):single

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
operator **(const x: Tvector2_single; const y: Tvector2_single) : single
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.4.265 power(Tvector3_double,Tvector3_double):Double

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
operator **(const x: Tvector3_double; const y: Tvector3_double) : Double
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.4.266 power(Tvector3_extended,Tvector3_extended):extended

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
operator **(const x: Tvector3_extended; const y: Tvector3_extended)
  : extended
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.4.267 power(Tvector3_single,Tvector3_single):single

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
operator **(const x: Tvector3_single; const y: Tvector3_single) : single
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.4.268 power(Tvector4_double,Tvector4_double):Double

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
operator **(const x: Tvector4_double; const y: Tvector4_double) : Double
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.4.269 power(Tvector4_extended,Tvector4_extended):extended

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
operator **(const x: Tvector4_extended; const y: Tvector4_extended)
  : extended
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.4.270 power(Tvector4_single,Tvector4_single):single

Synopsis: Calculate the internal product of two vectors.

Declaration:
```
operator **(const x: Tvector4_single; const y: Tvector4_single) : single
```

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.4.271 subtract(Tmatrix2_double,Double):Tmatrix2_double

Synopsis: Subtract scalar to two-dimensional double precision matrix.

Declaration:
```
operator -(const m: Tmatrix2_double; const x: Double) : Tmatrix2_double
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.4.272 subtract(Tmatrix2_double,Tmatrix2_double):Tmatrix2_double

Synopsis: Subtract a two-dimensional double precision matrix from another.

Declaration:
```
operator -(const m1: Tmatrix2_double; const m2: Tmatrix2_double)
                : Tmatrix2_double
```

Visibility: default

Description: This operator allows you to subtract a two-dimensional double precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.4.273 subtract(Tmatrix2_extended,extended):Tmatrix2_extended

Synopsis: Add scalar to two-dimensional extended precision matrix.

Declaration:
```
operator -(const m: Tmatrix2_extended; const x: extended)
                : Tmatrix2_extended
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.4.274 subtract(Tmatrix2_extended,Tmatrix2_extended):Tmatrix2_extended

Synopsis: Subtract a two-dimensional extended precision matrix from another.

Declaration:
```
operator -(const m1: Tmatrix2_extended; const m2: Tmatrix2_extended)
                : Tmatrix2_extended
```

Visibility: default

Description: This operator allows you to subtract a two-dimensional extended precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.4.275 subtract(Tmatrix2_single,single):Tmatrix2_single

Synopsis: Subtract scalar to two-dimensional single precision matrix.

Declaration:
```
operator -(const m: Tmatrix2_single; const x: single) : Tmatrix2_single
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.4.276 subtract(Tmatrix2_single,Tmatrix2_single):Tmatrix2_single

Synopsis: Subtract a two-dimensional single precision matrix from another.

Declaration:
```
operator -(const m1: Tmatrix2_single; const m2: Tmatrix2_single)
                : Tmatrix2_single
```

Visibility: default

Description: This operator allows you to subtract a two-dimensional single precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.4.277 subtract(Tmatrix3_double,Double):Tmatrix3_double

Synopsis: Add scalar to three-dimensional double precision matrix.

Declaration:
```
operator -(const m: Tmatrix3_double; const x: Double) : Tmatrix3_double
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.4.278 subtract(Tmatrix3_double,Tmatrix3_double):Tmatrix3_double

Synopsis: Subtract a three-dimensional double precision matrix from another.

Declaration:
```
operator -(const m1: Tmatrix3_double; const m2: Tmatrix3_double)
                : Tmatrix3_double
```

Visibility: default

Description: This operator allows you to subtract a three-dimensional double precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.4.279 subtract(Tmatrix3_extended,extended):Tmatrix3_extended

Synopsis: Add scalar to three-dimensional extended precision matrix.

Declaration:
```
operator -(const m: Tmatrix3_extended; const x: extended)
                : Tmatrix3_extended
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.4.280 subtract(Tmatrix3_extended,Tmatrix3_extended):Tmatrix3_extended

Synopsis: Subtract a three-dimensional extended precision matrix from another.

Declaration:
```
operator -(const m1: Tmatrix3_extended; const m2: Tmatrix3_extended)
  : Tmatrix3_extended
```

Visibility: default

Description: This operator allows you to subtract a three-dimensional extended precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.4.281 subtract(Tmatrix3_single,single):Tmatrix3_single

Synopsis: Add scalar to three-dimensional single precision matrix.

Declaration:
```
operator -(const m: Tmatrix3_single; const x: single) : Tmatrix3_single
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.4.282 subtract(Tmatrix3_single,Tmatrix3_single):Tmatrix3_single

Synopsis: Subtract a three-dimensional single precision matrix from another.

Declaration:
```
operator -(const m1: Tmatrix3_single; const m2: Tmatrix3_single)
  : Tmatrix3_single
```

Visibility: default

Description: This operator allows you to subtract a three-dimensional single precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.4.283 subtract(Tmatrix4_double,Double):Tmatrix4_double

Synopsis: Add scalar to four-dimensional double precision matrix.

Declaration:
```
operator -(const m: Tmatrix4_double; const x: Double) : Tmatrix4_double
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.4.284 subtract(Tmatrix4_double,Tmatrix4_double):Tmatrix4_double

Synopsis: Subtract a four-dimensional double precision matrix from another.

Declaration:
```
operator -(const m1: Tmatrix4_double; const m2: Tmatrix4_double)
  : Tmatrix4_double
```

Visibility: default

Description: This operator allows you to subtract a four-dimensional double precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.4.285 subtract(Tmatrix4_extended,extended):Tmatrix4_extended

Synopsis: Add scalar to four-dimensional extended precision matrix.

Declaration:
```
operator -(const m: Tmatrix4_extended; const x: extended)
  : Tmatrix4_extended
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.4.286 subtract(Tmatrix4_extended,Tmatrix4_extended):Tmatrix4_extended

Synopsis: Subtract a four-dimensional extended precision matrix from another.

Declaration:
```
operator -(const m1: Tmatrix4_extended; const m2: Tmatrix4_extended)
  : Tmatrix4_extended
```

Visibility: default

Description: This operator allows you to subtract a four-dimensional extended precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.4.287 subtract(Tmatrix4_single,single):Tmatrix4_single

Synopsis: Add scalar to four-dimensional single precision matrix.

Declaration:
```
operator -(const m: Tmatrix4_single; const x: single) : Tmatrix4_single
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.4.288 subtract(Tmatrix4_single,Tmatrix4_single):Tmatrix4_single

Synopsis: Subtract a four-dimensional single precision matrix from another.

Declaration:
```
operator -(const m1: Tmatrix4_single; const m2: Tmatrix4_single)
  : Tmatrix4_single
```

Visibility: default

Description: This operator allows you to subtract a four-dimensional single precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.4.289 subtract(Tvector2_double,Double):Tvector2_double

Synopsis: Subtract scalar from two-dimensional double precision vector.

Declaration:
```
operator -(const x: Tvector2_double; y: Double) : Tvector2_double
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.4.290 subtract(Tvector2_double,Tvector2_double):Tvector2_double

Synopsis: Subtract two-dimensional double precision vectors from each other.

Declaration:
```
operator -(const x: Tvector2_double; const y: Tvector2_double)
                  : Tvector2_double
```

Visibility: default

Description: This operator allows you to subtract two two-dimensional vectors with double precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.4.291 subtract(Tvector2_extended,extended):Tvector2_extended

Synopsis: Subtract scalar from two-dimensional extended precision vector.

Declaration:
```
operator -(const x: Tvector2_extended; y: extended) : Tvector2_extended
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.4.292 subtract(Tvector2_extended,Tvector2_extended):Tvector2_extended

Synopsis: Subtract two-dimensional extended precision vectors from each other.

Declaration:
```
operator -(const x: Tvector2_extended; const y: Tvector2_extended)
                  : Tvector2_extended
```

Visibility: default

Description: This operator allows you to subtract two two-dimensional vectors with extended precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.4.293 subtract(Tvector2_single,single):Tvector2_single

Synopsis: Subtract scalar from two-dimensional single precision vector.

Declaration:
```
operator -(const x: Tvector2_single; y: single) : Tvector2_single
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.4.294 subtract(Tvector2_single,Tvector2_single):Tvector2_single

Synopsis: Subtract two-dimensional single precision vectors from each other.

Declaration:
```
operator -(const x: Tvector2_single; const y: Tvector2_single)
                  : Tvector2_single
```

Visibility: default

Description: This operator allows you to subtract two two-dimensional vectors with single precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.4.295 subtract(Tvector3_double,Double):Tvector3_double

Synopsis: Subtract scalar from three-dimensional double precision vector.

Declaration:
```
operator -(const x: Tvector3_double; y: Double) : Tvector3_double
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.4.296 subtract(Tvector3_double,Tvector3_double):Tvector3_double

Synopsis: Subtract three-dimensional double precision vectors from each other.

Declaration:
```
operator -(const x: Tvector3_double; const y: Tvector3_double)
                : Tvector3_double
```

Visibility: default

Description: This operator allows you to subtract two two-dimensional vectors with double precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.4.297 subtract(Tvector3_extended,extended):Tvector3_extended

Synopsis: Subtract scalar from three-dimensional extended precision vector.

Declaration:
```
operator -(const x: Tvector3_extended; y: extended) : Tvector3_extended
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.4.298 subtract(Tvector3_extended,Tvector3_extended):Tvector3_extended

Synopsis: Subtract three-dimensional extended precision vectors from each other.

Declaration:
```
operator -(const x: Tvector3_extended; const y: Tvector3_extended)
                : Tvector3_extended
```

Visibility: default

Description: This operator allows you to subtract two three-dimensional vectors with extended precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.4.299 subtract(Tvector3_single,single):Tvector3_single

Synopsis: Subtract scalar from three-dimensional single precision vector.

Declaration:
```
operator -(const x: Tvector3_single; y: single) : Tvector3_single
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.4.300 subtract(Tvector3_single,Tvector3_single):Tvector3_single

Synopsis: Subtract three-dimensional single precision vectors from each other.

Declaration:
```
operator -(const x: Tvector3_single; const y: Tvector3_single)
                : Tvector3_single
```

Visibility: default

Description: This operator allows you to subtract two three-dimensional vectors with single precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.4.301 subtract(Tvector4_double,Double):Tvector4_double

Synopsis: Subtract scalar from four-dimensional double precision vector.

Declaration:
```
operator -(const x: Tvector4_double; y: Double) : Tvector4_double
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.4.302 subtract(Tvector4_double,Tvector4_double):Tvector4_double

Synopsis: Subtract four-dimensional double precision vectors from each other.

Declaration:
```
operator -(const x: Tvector4_double; const y: Tvector4_double)
                : Tvector4_double
```

Visibility: default

Description: This operator allows you to subtract two four-dimensional vectors with double precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.4.303 subtract(Tvector4_extended,extended):Tvector4_extended

Synopsis: Subtract scalar from four-dimensional extended precision vector.

Declaration:
```
operator -(const x: Tvector4_extended; y: extended) : Tvector4_extended
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.4.304 subtract(Tvector4_extended,Tvector4_extended):Tvector4_extended

Synopsis: Subtract four-dimensional extended precision vectors from each other.

Declaration:
```
operator -(const x: Tvector4_extended; const y: Tvector4_extended)
                : Tvector4_extended
```

Visibility: default

Description: This operator allows you to subtract two four-dimensional vectors with extended precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.4.305 subtract(Tvector4_single,single):Tvector4_single

Synopsis: Subtract scalar from four-dimensional single precision vector.

Declaration:
```
operator -(const x: Tvector4_single; y: single) : Tvector4_single
```

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.4.306 subtract(Tvector4_single,Tvector4_single):Tvector4_single

Synopsis: Subtract four-dimensional single precision vectors from each other.

Declaration:
```
operator -(const x: Tvector4_single; const y: Tvector4_single)
                : Tvector4_single
```

Visibility: default

Description: This operator allows you to subtract two four-dimensional vectors with single precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.4.307 symmetricaldifference(Tvector3_double,Tvector3_double):Tvector3_double

Synopsis: Calculate the external product of two three-dimensional vectors.

Declaration:
```
operator ><(const x: Tvector3_double; const y: Tvector3_double)
                : Tvector3_double
```

Visibility: default

Description: This operator returns the external product of two three dimensional vector. It is a vector orthonormal to the two multiplied vectors. The length of that vector is equal to the surface area of a parallelogram with the two vectors as sides.

The external product is often used to get a vector orthonormal to two other vectors, but of a predefined length. In order to do so, the result vector from the external product, is divided by its length, and then multiplied by the desired size.

64.4.308 symmetricaldifference(Tvector3_extended,Tvector3_extended):Tvector3_exten

Synopsis: Calculate the external product of two three-dimensional vectors.

Declaration:
```
operator ><(const x: Tvector3_extended; const y: Tvector3_extended)
                : Tvector3_extended
```

Visibility: default

Description: This operator returns the external product of two three dimensional vector. It is a vector orthonormal to the two multiplied vectors. The length of that vector is equal to the surface area of a parallelogram with the two vectors as sides.

The external product is often used to get a vector orthonormal to two other vectors, but of a predefined length. In order to do so, the result vector from the external product, is divided by its length, and then multiplied by the desired size.

64.4.309 symmetricaldifference(Tvector3_single,Tvector3_single):Tvector3_single

Synopsis: Calculate the external product of two three-dimensional vectors.

Declaration:
```
operator ><(const x: Tvector3_single; const y: Tvector3_single)
                  : Tvector3_single
```

Visibility: default

Description: This operator returns the external product of two three dimensional vector. It is a vector orthonormal to the two multiplied vectors. The length of that vector is equal to the surface area of a parallelogram with the two vectors as sides.

The external product is often used to get a vector orthonormal to two other vectors, but of a predefined length. In order to do so, the result vector from the external product, is divided by its length, and then multiplied by the desired size.

64.5 Tmatrix2_double

64.5.1 Description

The `Tmatrix2_double` object provides a matrix of 2*2 double precision scalars.

64.5.2 Method overview

Page	Method	Description
1130	determinant	Calculates the determinant of the matrix.
1130	get_column	Returns the c-th column of the matrix as vector.
1130	get_row	Returns the r-th row of the matrix as vector.
1130	init	Initializes the matrix, setting its elements to the values passed to the constructor.
1129	init_identity	Initializes the matrix and sets its elements to the identity matrix.
1129	init_zero	Initializes the matrix and sets its elements to zero.
1131	inverse	Calculates the inverse of the matrix.
1130	set_column	Sets c-th column of the matrix with a vector.
1130	set_row	Sets r-th row of the matrix with a vector.
1131	transpose	Returns the transposition of the matrix.

64.5.3 Tmatrix2_double.init_zero

Synopsis: Initializes the matrix and sets its elements to zero.

Declaration:
```
constructor init_zero
```

Visibility: default

64.5.4 Tmatrix2_double.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration:
```
constructor init_identity
```

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.5.5 Tmatrix2_double.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration: `constructor init(aa: Double; ab: Double; ba: Double; bb: Double)`

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.5.6 Tmatrix2_double.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration: `function get_column(c: Byte) : Tvector2_double`

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

64.5.7 Tmatrix2_double.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector2_double`

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

64.5.8 Tmatrix2_double.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte; const v: Tvector2_double)`

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

64.5.9 Tmatrix2_double.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte; const v: Tvector2_double)`

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

64.5.10 Tmatrix2_double.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : Double`

Visibility: default

Description: Returns the determinant of the matrix.

64.5.11 Tmatrix2_double.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: Double) : Tmatrix2_double`

Visibility: default

Description: Tmatrix2_double.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter.

64.5.12 Tmatrix2_double.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix2_double`

Visibility: default

Description: Tmatrix2_double.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

64.6 Tmatrix2_extended

64.6.1 Description

The `Tmatrix2_extended` object provides a matrix of 2*2 extended precision scalars.

64.6.2 Method overview

Page	Method	Description
1133	determinant	Calculates the determinant of the matrix.
1132	get_column	Returns the c-th column of the matrix as vector.
1132	get_row	Returns the r-th row of the matrix as vector.
1132	init	Initializes the matrix, setting its elements to the values passed to the constructor.
1132	init_identity	Initializes the matrix and sets its elements to the identity matrix.
1131	init_zero	Initializes the matrix and sets its elements to zero.
1133	inverse	Calculates the inverse of the matrix.
1132	set_column	Sets c-th column of the matrix with a vector.
1132	set_row	Sets r-th row of the matrix with a vector.
1133	transpose	Returns the transposition of the matrix.

64.6.3 Tmatrix2_extended.init_zero

Synopsis: Initializes the matrix and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.6.4 Tmatrix2_extended.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration: `constructor init_identity`

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.6.5 Tmatrix2_extended.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration: `constructor init(aa: extended; ab: extended; ba: extended; bb: extended)`

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.6.6 Tmatrix2_extended.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration: `function get_column(c: Byte) : Tvector2_extended`

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

64.6.7 Tmatrix2_extended.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector2_extended`

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

64.6.8 Tmatrix2_extended.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte; const v: Tvector2_extended)`

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

64.6.9 Tmatrix2_extended.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte; const v: Tvector2_extended)`

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

64.6.10 Tmatrix2_extended.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : extended`

Visibility: default

Description: Returns the determinant of the matrix.

64.6.11 Tmatrix2_extended.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: extended) : Tmatrix2_extended`

Visibility: default

Description: Tmatrix2_extended.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter.

64.6.12 Tmatrix2_extended.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix2_extended`

Visibility: default

Description: Tmatrix2_extended.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

64.7 Tmatrix2_single

64.7.1 Description

The `Tmatrix2_single` object provides a matrix of 2*2 single precision scalars.

64.7.2 Method overview

Page	Method	Description
1135	determinant	Calculates the determinant of the matrix.
1134	get_column	Returns the c-th column of the matrix as vector.
1134	get_row	Returns the r-th row of the matrix as vector.
1134	init	Initializes the matrix, setting its elements to the values passed to the constructor.
1134	init_identity	Initializes the matrix and sets its elements to the identity matrix.
1134	init_zero	Initializes the matrix and sets its elements to zero.
1135	inverse	Calculates the inverse of the matrix.
1134	set_column	Sets c-th column of the matrix with a vector.
1135	set_row	Sets r-th row of the matrix with a vector.
1135	transpose	Returns the transposition of the matrix.

64.7.3 Tmatrix2_single.init_zero

Synopsis: Initializes the matrix and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.7.4 Tmatrix2_single.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration: `constructor init_identity`

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.7.5 Tmatrix2_single.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration: `constructor init(aa: single; ab: single; ba: single; bb: single)`

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.7.6 Tmatrix2_single.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration: `function get_column(c: Byte) : Tvector2_single`

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

64.7.7 Tmatrix2_single.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector2_single`

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

64.7.8 Tmatrix2_single.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte; const v: Tvector2_single)`

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

64.7.9 Tmatrix2_single.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte; const v: Tvector2_single)`

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

64.7.10 Tmatrix2_single.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : single`

Visibility: default

Description: Returns the determinant of the matrix.

64.7.11 Tmatrix2_single.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: single) : Tmatrix2_single`

Visibility: default

Description: Tmatrix2_single.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter.

64.7.12 Tmatrix2_single.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix2_single`

Visibility: default

Description: Tmatrix2_single.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

64.8 Tmatrix3_double

64.8.1 Description

The `Tmatrix3_double` object provides a matrix of 3*3 double precision scalars.

64.8.2 Method overview

Page	Method	Description
1137	determinant	Calculates the determinant of the matrix.
1136	get_column	Returns the c-th column of the matrix as vector.
1137	get_row	Returns the r-th row of the matrix as vector.
1136	init	Initializes the matrix, setting its elements to the values passed to the constructor.
1136	init_identity	Initializes the matrix and sets its elements to the identity matrix.
1136	init_zero	Initializes the matrix and sets its elements to zero.
1137	inverse	Calculates the inverse of the matrix.
1137	set_column	Sets c-th column of the matrix with a vector.
1137	set_row	Sets r-th row of the matrix with a vector.
1137	transpose	Returns the transposition of the matrix.

64.8.3 Tmatrix3_double.init_zero

Synopsis: Initializes the matrix and sets its elements to zero.

Declaration:
```
constructor init_zero
```

Visibility: default

64.8.4 Tmatrix3_double.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration:
```
constructor init_identity
```

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.8.5 Tmatrix3_double.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration:
```
constructor init(aa: Double; ab: Double; ac: Double; ba: Double;
                 bb: Double; bc: Double; ca: Double; cb: Double;
                 cc: Double)
```

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.8.6 Tmatrix3_double.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration:
```
function get_column(c: Byte) : Tvector3_double
```

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

64.8.7 Tmatrix3_double.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector3_double`

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

64.8.8 Tmatrix3_double.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte; const v: Tvector3_double)`

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

64.8.9 Tmatrix3_double.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte; const v: Tvector3_double)`

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

64.8.10 Tmatrix3_double.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : Double`

Visibility: default

Description: Returns the determinant of the matrix.

64.8.11 Tmatrix3_double.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: Double) : Tmatrix3_double`

Visibility: default

Description: Tmatrix3_double.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter.

64.8.12 Tmatrix3_double.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix3_double`

Visibility: default

Description: Tmatrix2_double.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

64.9 Tmatrix3_extended

64.9.1 Description

The `Tmatrix3_extended` object provides a matrix of 3*3 extended precision scalars.

64.9.2 Method overview

Page	Method	Description
1139	determinant	Calculates the determinant of the matrix.
1139	get_column	Returns the c-th column of the matrix as vector.
1139	get_row	Returns the r-th row of the matrix as vector.
1138	init	Initializes the matrix, setting its elements to the values passed to the constructor.
1138	init_identity	Initializes the matrix and sets its elements to the identity matrix.
1138	init_zero	Initializes the matrix and sets its elements to zero.
1139	inverse	Calculates the inverse of the matrix.
1139	set_column	Sets r-th column of the matrix with a vector.
1139	set_row	Sets r-th row of the matrix with a vector.
1140	transpose	Returns the transposition of the matrix.

64.9.3 Tmatrix3_extended.init_zero

Synopsis: Initializes the matrix and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.9.4 Tmatrix3_extended.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration: `constructor init_identity`

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.9.5 Tmatrix3_extended.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration:
```
constructor init(aa: extended; ab: extended; ac: extended;
                 ba: extended; bb: extended; bc: extended; ca: extended;
                 cb: extended; cc: extended)
```

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.9.6 Tmatrix3_extended.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration: `function get_column(c: Byte) : Tvector3_extended`

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

64.9.7 Tmatrix3_extended.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector3_extended`

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

64.9.8 Tmatrix3_extended.set_column

Synopsis: Sets r-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte; const v: Tvector3_extended)`

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

64.9.9 Tmatrix3_extended.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte; const v: Tvector3_extended)`

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

64.9.10 Tmatrix3_extended.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : extended`

Visibility: default

Description: Returns the determinant of the matrix.

64.9.11 Tmatrix3_extended.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: extended) : Tmatrix3_extended`

Visibility: default

Description: Tmatrix3_extended.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter.

64.9.12 Tmatrix3_extended.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix3_extended`

Visibility: default

Description: Tmatrix2_extended.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

64.10 Tmatrix3_single

64.10.1 Description

The `Tmatrix3_single` object provides a matrix of 3*3 single precision scalars.

64.10.2 Method overview

Page	Method	Description
1142	determinant	Calculates the determinant of the matrix.
1141	get_column	Returns the c-th column of the matrix as vector.
1141	get_row	Returns the r-th row of the matrix as vector.
1141	init	Initializes the matrix, setting its elements to the values passed to the constructor.
1140	init_identity	Initializes the matrix and sets its elements to the identity matrix.
1140	init_zero	Initializes the matrix and sets its elements to zero.
1142	inverse	Calculates the inverse of the matrix.
1141	set_column	Sets c-th column of the matrix with a vector.
1141	set_row	Sets r-th row of the matrix with a vector.
1142	transpose	Returns the transposition of the matrix.

64.10.3 Tmatrix3_single.init_zero

Synopsis: Initializes the matrix and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.10.4 Tmatrix3_single.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration: `constructor init_identity`

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.10.5 Tmatrix3_single.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration:
```
constructor init(aa: single; ab: single; ac: single; ba: single;
                 bb: single; bc: single; ca: single; cb: single;
                 cc: single)
```

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.10.6 Tmatrix3_single.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration:
```
function get_column(c: Byte) : Tvector3_single
```

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

64.10.7 Tmatrix3_single.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration:
```
function get_row(r: Byte) : Tvector3_single
```

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

64.10.8 Tmatrix3_single.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration:
```
procedure set_column(c: Byte; const v: Tvector3_single)
```

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

64.10.9 Tmatrix3_single.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration:
```
procedure set_row(r: Byte; const v: Tvector3_single)
```

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

64.10.10 Tmatrix3_single.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : single`

Visibility: default

Description: Returns the determinant of the matrix.

64.10.11 Tmatrix3_single.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: single) : Tmatrix3_single`

Visibility: default

Description: Tmatrix3_single.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter.

64.10.12 Tmatrix3_single.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix3_single`

Visibility: default

Description: Tmatrix2_single.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

64.11 Tmatrix4_double

64.11.1 Description

The `Tmatrix4_double` object provides a matrix of 4*4 double precision scalars.

64.11.2 Method overview

Page	Method	Description
1144	determinant	Calculates the determinant of the matrix.
1143	get_column	Returns the c-th column of the matrix as vector.
1143	get_row	Returns the r-th row of the matrix as vector.
1143	init	Initializes the matrix, setting its elements to the values passed to the constructor.
1143	init_identity	Initializes the matrix and sets its elements to the identity matrix.
1143	init_zero	Initializes the matrix and sets its elements to zero.
1144	inverse	Calculates the inverse of the matrix.
1144	set_column	Sets c-th column of the matrix with a vector.
1144	set_row	Sets r-th row of the matrix with a vector.
1144	transpose	Returns the transposition of the matrix.

64.11.3 Tmatrix4_double.init_zero

Synopsis: Initializes the matrix and sets its elements to zero.

Declaration:
```
constructor init_zero
```

Visibility: default

64.11.4 Tmatrix4_double.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration:
```
constructor init_identity
```

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.11.5 Tmatrix4_double.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration:
```
constructor init(aa: Double; ab: Double; ac: Double; ad: Double;
                 ba: Double; bb: Double; bc: Double; bd: Double;
                 ca: Double; cb: Double; cc: Double; cd: Double;
                 da: Double; db: Double; dc: Double; dd: Double)
```

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.11.6 Tmatrix4_double.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration:
```
function get_column(c: Byte) : Tvector4_double
```

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

64.11.7 Tmatrix4_double.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration:
```
function get_row(r: Byte) : Tvector4_double
```

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

64.11.8 Tmatrix4_double.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte; const v: Tvector4_double)`

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

64.11.9 Tmatrix4_double.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte; const v: Tvector4_double)`

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

64.11.10 Tmatrix4_double.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : Double`

Visibility: default

Description: Returns the determinant of the matrix. Note: Calculating the determinant of a 4*4 matrix requires quite a few operations.

64.11.11 Tmatrix4_double.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: Double) : Tmatrix4_double`

Visibility: default

Description: Tmatrix4_double.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter. Note: Calculating the inverse of a 4*4 matrix requires quite a few operations.

64.11.12 Tmatrix4_double.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix4_double`

Visibility: default

Description: Tmatrix2_double.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

64.12 Tmatrix4_extended

64.12.1 Description

The `Tmatrix4_extended` object provides a matrix of 4*4 extended precision scalars.

64.12.2 Method overview

Page	Method	Description
1146	determinant	Calculates the determinant of the matrix.
1146	get_column	Returns the c-th column of the matrix as vector.
1146	get_row	Returns the r-th row of the matrix as vector.
1145	init	Initializes the matrix, setting its elements to the values passed to the constructor.
1145	init_identity	Initializes the matrix and sets its elements to the identity matrix.
1145	init_zero	Initializes the matrix and sets its elements to zero.
1147	inverse	Calculates the inverse of the matrix.
1146	set_column	Sets c-th column of the matrix with a vector.
1146	set_row	Sets r-th row of the matrix with a vector.
1147	transpose	Returns the transposition of the matrix.

64.12.3 Tmatrix4_extended.init_zero

Synopsis: Initializes the matrix and sets its elements to zero.

Declaration:
```
constructor init_zero
```

Visibility: default

64.12.4 Tmatrix4_extended.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration:
```
constructor init_identity
```

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.12.5 Tmatrix4_extended.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration:
```
constructor init(aa: extended; ab: extended; ac: extended;
                 ad: extended; ba: extended; bb: extended; bc: extended;
                 bd: extended; ca: extended; cb: extended; cc: extended;
                 cd: extended; da: extended; db: extended; dc: extended;
                 dd: extended)
```

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.12.6 Tmatrix4_extended.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration: `function get_column(c: Byte) : Tvector4_extended`

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

64.12.7 Tmatrix4_extended.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector4_extended`

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

64.12.8 Tmatrix4_extended.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte; const v: Tvector4_extended)`

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

64.12.9 Tmatrix4_extended.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte; const v: Tvector4_extended)`

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

64.12.10 Tmatrix4_extended.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : extended`

Visibility: default

Description: Returns the determinant of the matrix. Note: Calculating the determinant of a 4*4 matrix requires quite a few operations.

64.12.11 Tmatrix4_extended.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: extended) : Tmatrix4_extended`

Visibility: default

Description: Tmatrix4_extended.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter. Note: Calculating the inverse of a 4*4 matrix requires quite a few operations.

64.12.12 Tmatrix4_extended.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix4_extended`

Visibility: default

Description: Tmatrix2_extended.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

64.13 Tmatrix4_single

64.13.1 Description

The `Tmatrix4_single` object provides a matrix of 4*4 single precision scalars.

64.13.2 Method overview

Page	Method	Description
1149	determinant	Calculates the determinant of the matrix.
1148	get_column	Returns the c-th column of the matrix as vector.
1148	get_row	Returns the r-th row of the matrix as vector.
1148	init	Initializes the matrix, setting its elements to the values passed to the constructor.
1148	init_identity	Initializes the matrix and sets its elements to the identity matrix.
1147	init_zero	Initializes the matrix and sets its elements to zero.
1149	inverse	Calculates the inverse of the matrix.
1148	set_column	Sets c-th column of the matrix with a vector.
1149	set_row	Sets r-th row of the matrix with a vector.
1149	transpose	Returns the transposition of the matrix.

64.13.3 Tmatrix4_single.init_zero

Synopsis: Initializes the matrix and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.13.4 Tmatrix4_single.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration:
```
constructor init_identity
```

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.13.5 Tmatrix4_single.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration:
```
constructor init(aa: single; ab: single; ac: single; ad: single;
                 ba: single; bb: single; bc: single; bd: single;
                 ca: single; cb: single; cc: single; cd: single;
                 da: single; db: single; dc: single; dd: single)
```

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.13.6 Tmatrix4_single.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration:
```
function get_column(c: Byte) : Tvector4_single
```

Visibility: default

Description: Returns the `c`-th column of the matrix as vector. The column numbering starts at 0.

64.13.7 Tmatrix4_single.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration:
```
function get_row(r: Byte) : Tvector4_single
```

Visibility: default

Description: Returns the `r`-th row of the matrix as vector. The row numbering starts at 0.

64.13.8 Tmatrix4_single.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration:
```
procedure set_column(c: Byte; const v: Tvector4_single)
```

Visibility: default

Description: Replaces the `c`-th column of the matrix with vector v. The column numbering starts at 0.

64.13.9 Tmatrix4_single.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte; const v: Tvector4_single)`

Visibility: default

Description: Replaces the `r`-th row of the matrix with vector v. The row numbering starts at 0.

64.13.10 Tmatrix4_single.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : single`

Visibility: default

Description: Returns the determinant of the matrix. Note: Calculating the determinant of a 4*4 matrix requires quite a few operations.

64.13.11 Tmatrix4_single.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: single) : Tmatrix4_single`

Visibility: default

Description: Tmatrix4_single.inverse returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter. Note: Calculating the inverse of a 4*4 matrix requires quite a few operations.

64.13.12 Tmatrix4_single.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix4_single`

Visibility: default

Description: Tmatrix2_single.transpose returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

64.14 Tvector2_double

64.14.1 Description

The `Tvector2_double` object provides a vector of two double precision scalars.

64.14.2 Method overview

Page	Method	Description
1150	init	Initializes the vector, setting its elements to the values passed to the constructor.
1150	init_one	Initializes the vector and sets its elements to one.
1150	init_zero	Initializes the vector and sets its elements to zero.
1150	length	Calculates the length of the vector.
1150	squared_length	Calculates the squared length of the vector.

64.14.3 Tvector2_double.init_zero

Synopsis: Initializes the vector and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.14.4 Tvector2_double.init_one

Synopsis: Initializes the vector and sets its elements to one.

Declaration: `constructor init_one`

Visibility: default

64.14.5 Tvector2_double.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: Double; b: Double)`

Visibility: default

64.14.6 Tvector2_double.length

Synopsis: Calculates the length of the vector.

Declaration: `function &length : Double`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2). Try to use squared_length (1150) if you are able to, as it is faster.

64.14.7 Tvector2_double.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : Double`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2.

64.15 Tvector2_extended

64.15.1 Description

The `Tvector2_extended` object provides a vector of two extended precision scalars.

64.15.2 Method overview

Page	Method	Description
1151	init	Initializes the vector, setting its elements to the values passed to the constructor.
1151	init_one	Initializes the vector and sets its elements to one.
1151	init_zero	Initializes the vector and sets its elements to zero.
1151	length	Calculates the length of the vector.
1152	squared_length	Calculates the squared length of the vector.

64.15.3 Tvector2_extended.init_zero

Synopsis: Initializes the vector and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.15.4 Tvector2_extended.init_one

Synopsis: Initializes the vector and sets its elements to one.

Declaration: `constructor init_one`

Visibility: default

64.15.5 Tvector2_extended.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: extended; b: extended)`

Visibility: default

64.15.6 Tvector2_extended.length

Synopsis: Calculates the length of the vector.

Declaration: `function &length : extended`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2). Try to use squared_length (1152) if you are able to, as it is faster.

64.15.7 Tvector2_extended.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : extended`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2.

64.16 Tvector2_single

64.16.1 Description

The `Tvector2_single` object provides a vector of two single precision scalars.

64.16.2 Method overview

Page	Method	Description
1152	init	Initializes the vector, setting its elements to the values passed to the constructor.
1152	init_one	Initializes the vector and sets its elements to one.
1152	init_zero	Initializes the vector and sets its elements to zero.
1153	length	Calculates the length of the vector.
1153	squared_length	Calculates the squared length of the vector.

64.16.3 Tvector2_single.init_zero

Synopsis: Initializes the vector and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.16.4 Tvector2_single.init_one

Synopsis: Initializes the vector and sets its elements to one.

Declaration: `constructor init_one`

Visibility: default

64.16.5 Tvector2_single.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: single; b: single)`

Visibility: default

64.16.6 Tvector2_single.length

Synopsis: Calculates the length of the vector.

Declaration: `function &length : single`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2). Try to use squared_length (1153) if you are able to, as it is faster.

64.16.7 Tvector2_single.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : single`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2.

64.17 Tvector3_double

64.17.1 Description

The `Tvector3_double` object provides a vector of three double precision scalars.

64.17.2 Method overview

Page	Method	Description
1154	init	Initializes the vector, setting its elements to the values passed to the constructor.
1153	init_one	Initializes the vector and sets its elements to one.
1153	init_zero	Initializes the vector and sets its elements to zero.
1154	length	Calculates the length of the vector.
1154	squared_length	Calculates the squared length of the vector.

64.17.3 Tvector3_double.init_zero

Synopsis: Initializes the vector and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.17.4 Tvector3_double.init_one

Synopsis: Initializes the vector and sets its elements to one.

Declaration: `constructor init_one`

Visibility: default

64.17.5 Tvector3_double.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: Double; b: Double; c: Double)`

Visibility: default

64.17.6 Tvector3_double.length

Synopsis: Calculates the length of the vector.

Declaration: `function &length : Double`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2+data[2]**2). Try to use squared_length (1154) if you are able to, as it is faster.

64.17.7 Tvector3_double.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : Double`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2+data[2]**2.

64.18 Tvector3_extended

64.18.1 Description

The `Tvector3_extended` object provides a vector of three extended precision scalars.

64.18.2 Method overview

Page	Method	Description
1155	init	Initializes the vector, setting its elements to the values passed to the constructor.
1155	init_one	Initializes the vector and sets its elements to one.
1154	init_zero	Initializes the vector and sets its elements to zero.
1155	length	Calculates the length of the vector.
1155	squared_length	Calculates the squared length of the vector.

64.18.3 Tvector3_extended.init_zero

Synopsis: Initializes the vector and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.18.4 Tvector3_extended.init_one

Synopsis: Initializes the vector and sets its elements to one.

Declaration: `constructor init_one`

Visibility: default

64.18.5 Tvector3_extended.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: extended; b: extended; c: extended)`

Visibility: default

64.18.6 Tvector3_extended.length

Synopsis: Calculates the length of the vector.

Declaration: `function &length : extended`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2+data[2]**2). Try to use squared_length (1155) if you are able to, as it is faster.

64.18.7 Tvector3_extended.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : extended`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2+data[2]**2.

64.19 Tvector3_single

64.19.1 Description

The `Tvector3_single` object provides a vector of three single precision scalars.

64.19.2 Method overview

Page	Method	Description
1156	init	Initializes the vector, setting its elements to the values passed to the constructor.
1156	init_one	Initializes the vector and sets its elements to one.
1156	init_zero	Initializes the vector and sets its elements to zero.
1156	length	Calculates the length of the vector.
1156	squared_length	Calculates the squared length of the vector.

64.19.3 Tvector3_single.init_zero

Synopsis: Initializes the vector and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.19.4 Tvector3_single.init_one

Synopsis: Initializes the vector and sets its elements to one.

Declaration: `constructor init_one`

Visibility: default

64.19.5 Tvector3_single.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: single; b: single; c: single)`

Visibility: default

64.19.6 Tvector3_single.length

Synopsis: Calculates the length of the vector.

Declaration: `function &length : single`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2+data[2]**2). Try to use squared_length (1156) if you are able to, as it is faster.

64.19.7 Tvector3_single.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : single`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2+data[2]**2.

64.20 Tvector4_double

64.20.1 Description

The `Tvector4_double` object provides a vector of four double precision scalars.

64.20.2 Method overview

Page	Method	Description
1157	init	Initializes the vector, setting its elements to the values passed to the constructor.
1157	init_one	Initializes the vector and sets its elements to one.
1157	init_zero	Initializes the vector and sets its elements to zero.
1157	length	Calculates the length of the vector.
1157	squared_length	Calculates the squared length of the vector.

64.20.3 Tvector4_double.init_zero

Synopsis: Initializes the vector and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.20.4 Tvector4_double.init_one

Synopsis: Initializes the vector and sets its elements to one.

Declaration: `constructor init_one`

Visibility: default

64.20.5 Tvector4_double.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: Double; b: Double; c: Double; d: Double)`

Visibility: default

64.20.6 Tvector4_double.length

Synopsis: Calculates the length of the vector.

Declaration: `function &length : Double`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2+data[2]**2+data[3]**2). Try to use squared_length (1157) if you are able to, as it is faster.

64.20.7 Tvector4_double.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : Double`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2+data[2]**2+data[3]**2.

64.21 Tvector4_extended

64.21.1 Description

The `Tvector4_extended` object provides a vector of four extended precision scalars.

64.21.2 Method overview

Page	Method	Description
1158	init	Initializes the vector, setting its elements to the values passed to the constructor.
1158	init_one	Initializes the vector and sets its elements to one.
1158	init_zero	Initializes the vector and sets its elements to zero.
1158	length	Calculates the length of the vector.
1159	squared_length	Calculates the squared length of the vector.

64.21.3 Tvector4_extended.init_zero

Synopsis: Initializes the vector and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.21.4 Tvector4_extended.init_one

Synopsis: Initializes the vector and sets its elements to one.

Declaration: `constructor init_one`

Visibility: default

64.21.5 Tvector4_extended.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: extended; b: extended; c: extended; d: extended)`

Visibility: default

64.21.6 Tvector4_extended.length

Synopsis: Calculates the length of the vector.

Declaration: `function &length : extended`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2+data[2]**2+data[3]**2). Try to use squared_length (1159) if you are able to, as it is faster.

64.21.7 Tvector4_extended.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : extended`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2+data[2]**2+data[3]**2.

64.22 Tvector4_single

64.22.1 Description

The `Tvector4_single` object provides a vector of four single precision scalars.

64.22.2 Method overview

Page	Method	Description
1159	init	Initializes the vector, setting its elements to the values passed to the constructor.
1159	init_one	Initializes the vector and sets its elements to one.
1159	init_zero	Initializes the vector and sets its elements to zero.
1160	length	Calculates the length of the vector.
1160	squared_length	Calculates the squared length of the vector.

64.22.3 Tvector4_single.init_zero

Synopsis: Initializes the vector and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.22.4 Tvector4_single.init_one

Synopsis: Initializes the vector and sets its elements to one.

Declaration: `constructor init_one`

Visibility: default

64.22.5 Tvector4_single.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: single; b: single; c: single; d: single)`

Visibility: default

64.22.6 Tvector4_single.length

Synopsis: Calculates the length of the vector.

Declaration: `function &length : single`

Visibility: default

Description: Calculate the length of the vector: length=sqrt(data[0]**2+data[1]**2+data[2]**2+data[3]**2). Try to use squared_length (1160) if you are able to, as it is faster.

64.22.7 Tvector4_single.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : single`

Visibility: default

Description: Calculate the squared length of the vector: squared_length=data[0]**2+data[1]**2+data[2]**2+data[3]**2.

Chapter 65

Reference for unit 'mmx'

65.1 Used units

Table 65.1: Used units by unit 'mmx'

Name	Page
System	1362

65.2 Overview

This document describes the MMX unit. This unit allows you to use the MMX capabilities of the Free Pascal compiler. It was written by Florian Klaempfl for the I386 processor. It should work on all platforms that use the Intel processor.

65.3 Constants, types and variables

65.3.1 Constants

```
is_amd_3d_cpu : Boolean = False
```

The is_amd_3d_cpu initialized constant allows you to determine if the computer has the AMD 3D extensions. It is set correctly in the unit's initialization code.

```
is_amd_3d_dsp_cpu : Boolean = False
```

The is_amd_3d_dsp_cpu initialized constant allows you to determine if the computer has the AMD 3D DSP extensions. It is set correctly in the unit's initialization code.

```
is_amd_3d_mmx_cpu : Boolean = False
```

The is_amd_3d_mmx_cpu initialized constant allows you to determine if the computer has the AMD 3D MMX extensions. It is set correctly in the unit's initialization code.

```
is_mmx_cpu : Boolean = False
```

The is_mmx_cpu initialized constant allows you to determine if the computer has MMX extensions. It is set correctly in the unit's initialization code.

```
is_sse2_cpu : Boolean = False
```

The is_sse2_cpu initialized constant allows you to determine if the computer has the SSE2 extensions. It is set correctly in the unit's initialization code.

```
is_sse_cpu : Boolean = False
```

The is_sse_cpu initialized constant allows you to determine if the computer has the SSE extensions. It is set correctly in the unit's initialization code.

65.3.2 Types

```
pmmxbyte = ^tmmxbyte
```

Pointer to tmmxbyte (1162) array type.

```
pmmxcardinal = ^tmmxcardinal
```

Pointer to tmmxcardinal (1162) array type.

```
pmmxinteger = ^tmmxinteger
```

Pointer to tmmxinteger (1163) array type.

```
pmmxlongint = ^tmmxlongint
```

Pointer to tmmxlongint (1163) array type.

```
pmmxshortint = ^tmmxshortint
```

Pointer to tmmxshortint (1163) array type.

```
pmmxsingle = ^tmmxsingle
```

Pointer to tmmxsingle (1163) array type.

```
pmmxword = ^tmmxword
```

Pointer to tmmxword (1163) array type.

```
tmmxbyte = Array[0..7] of Byte
```

Array of bytes, 64 bits in size.

```
tmmxcardinal = Array[0..1] of Cardinal
```

Array of cardinals, 64 bits in size.

```
tmmxinteger = Array[0..3] of Integer
```

Array of integers, 64 bits in size.

```
tmmxlongint = Array[0..1] of LongInt
```

Array of longint, 64 bits in size.

```
tmmxshortint = Array[0..7] of ShortInt
```

Array of shortints, 64 bits in size.

```
tmmxsingle = Array[0..1] of single
```

Array of singles, 64 bits in size.

```
tmmxword = Array[0..3] of Word
```

Array of words, 64 bits in size.

65.4 Procedures and functions

65.4.1 emms

Synopsis: Reset floating point registers.

Declaration: `procedure emms`

Visibility: default

Description: `Emms` sets all floating point registers to empty. This procedure must be called after you have used any `MMX` instructions, if you want to use floating point arithmetic. If you just want to move floating point data around, it isn't necessary to call this function, the compiler doesn't use the FPU registers when moving data. Only when doing calculations, you should use this function. The following code demonstrates this:

```
Program MMXDemo;
uses mmx;
var
   d : double;
   a : array[0..10000] of double;
   i : longint;
begin
   d:=1.0;
{$mmx+}
   { floating point data is used, but we do _no_ arithmetic }
   for i:=0 to 10000 do
     a[i]:=d;  { this is done with 64 bit moves }
{$mmx-}
   emms;   { clear fpu }
   { now we can do floating point arithmetic again }
end.
```

See also: femms (1164)

65.4.2 femms

Synopsis: Reset floating point registers - AMD version.

Declaration: `procedure femms`

Visibility: default

Description: `femms` executes the `femms` assembler instruction for AMD processors. it is not supported by all assemblers, hence it is coded as byte codes.

See also: emms (1163)

Chapter 66

Reference for unit 'Mouse'

66.1 Used units

Table 66.1: Used units by unit 'Mouse'

Name	Page
System	1362

66.2 Overview

The Mouse unit implements a platform independent mouse handling interface. It is implemented identically on all platforms supported by Free Pascal and can be enhanced with custom drivers, should this be needed. It is intended to be used only in text-based screens, for instance in conjunction with the keyboard and video unit. No support for graphical screens is implemented, and there are (currently) no plans to implement this.

66.3 Writing a custom mouse driver.

The mouse unit has support for adding a custom mouse driver. This can be used to add support for mouses not supported by the standard Free Pascal driver, but also to enhance an existing driver for instance to log mouse events or to implement a record and playback function.

The following unit shows how a mouse driver can be enhanced by adding some logging capabilities to the driver.

Listing: ./examples/mouseex/logmouse.pp

```
unit logmouse;

interface

Procedure StartMouseLogging;
Procedure StopMouseLogging;
Function  IsMouseLogging : Boolean;
Procedure SetMouseLogFileName(FileName : String);
```

```
implementation

uses sysutils,Mouse;

var
  NewMouseDriver,
  OldMouseDriver : TMouseDriver;
  Active,Logging : Boolean;
  LogFileName : String;
  MouseLog : Text;

Function TimeStamp : String;

begin
  TimeStamp:=FormatDateTime('hh:nn:ss',Time());
end;

Procedure StartMouseLogging;

begin
  Logging:=True;
  Writeln(MouseLog,'Start logging mouse events at: ',TimeStamp);
end;

Procedure StopMouseLogging;

begin
  Writeln(MouseLog,'Stop logging mouse events at: ',TimeStamp);
  Logging:=False;
end;

Function IsMouseLogging : Boolean;

begin
  IsMouseLogging:=Logging;
end;

Procedure LogGetMouseEvent(Var Event : TMouseEvent);

Var
  M : TMouseEvent;

begin
  OldMouseDriver.GetMouseEvent(M);
  If Logging then
    begin
    Write(MouseLog,TimeStamp,': Mouse ');
    With M do
      begin
      Case Action of
        MouseActionDown : Write(MouseLog,'down');
        MouseActionUp   : Write(MouseLog,'up');
        MouseActionMove : Write(MouseLog,'move');
      end;
      Write(MouseLog,' event at ',X,',',Y);
      If (Buttons<>0) then
        begin
```

```
          Write(MouseLog,' for buttons: ');
          If (Buttons and MouseLeftbutton)<>0 then
            Write(MouseLog,'Left ');
          If (Buttons and MouseRightbutton)<>0 then
            Write(MouseLog,'Right ');
          If (Buttons and MouseMiddlebutton)<>0 then
            Write(MouseLog,'Middle ');
         end;
        WriteIn(MouseLog);
       end;
     end;
end;

Procedure LogInitMouse;

begin
  OldMouseDriver.InitDriver();
  Assign(MouseLog,logFileName);
  Rewrite(MouseLog);
  Active:=True;
  StartMouseLogging;
end;

Procedure LogDoneMouse;

begin
  StopMouseLogging;
  Close(MouseLog);
  Active:=False;
  OldMouseDriver.DoneDriver();
end;

Procedure SetMouseLogFileName(FileName : String);

begin
  If Not Active then
    LogFileName:=FileName;
end;

Initialization
  GetMouseDriver(OldMouseDriver);
  NewMouseDriver:=OldMouseDriver;
  NewMouseDriver.GetMouseEvent:=@LogGetMouseEvent;
  NewMouseDriver.InitDriver:=@LogInitMouse;
  NewMouseDriver.DoneDriver:=@LogDoneMouse;
  LogFileName:='Mouse.log';
  Logging:=False;
  SetMouseDriver(NewMouseDriver);
end.
```

66.4 Constants, types and variables

66.4.1 Constants

```
errMouseBase = 1030
```

Base for mouse error codes.

```
errMouseInitError = errMouseBase + 0
```

Mouse initialization error.

```
errMouseNotImplemented = errMouseBase + 1
```

Mouse driver not implemented.

```
MouseActionDown = $0001
```

Mouse button down event signal.

```
MouseActionMove = $0004
```

Mouse cursor move event signal.

```
MouseActionUp = $0002
```

Mouse button up event signal.

```
MouseButton4 = $08
```

4th mouse button event.

```
MouseButton5 = $10
```

5th mouse button event.

```
MouseEventBufSize = 16
```

The mouse unit has a mechanism to buffer mouse events. This constant defines the size of the event buffer.

```
MouseLeftButton = $01
```

Left mouse button event.

```
MouseMiddleButton = $04
```

Middle mouse button event.

```
MouseRightButton = $02
```

Right mouse button event.

66.4.2 Types

```
PMouseEvent = ^TMouseEvent
```

Pointer to TMouseEvent (1176) record.

66.4.3 Variables

`MouseButtons : Byte`

This variable keeps track of the last known mouse button state. Do not use.

`MouseIntFlag : Byte`

This variable keeps track of the last known internal mouse state. Do not use.

`MouseWhereX : Word`

This variable keeps track of the last known cursor position. Do not use.

`MouseWhereY : Word`

This variable keeps track of the last known cursor position. Do not use.

66.5 Procedures and functions

66.5.1 DetectMouse

Synopsis: Detect the presence of a mouse.

Declaration: `function DetectMouse : Byte`

Visibility: default

Description: `DetectMouse` detects whether a mouse is attached to the system or not. If there is no mouse, then zero is returned. If a mouse is attached, then the number of mouse buttons is returned.

This function should be called after the mouse driver was initialized.

Errors: None.

See also: InitMouse (1173), DoneMouse (1170)

Listing: ./examples/mouseex/ex1.pp

```
Program Example1;

{ Program to demonstrate the DetectMouse function. }

Uses mouse;

Var
  Buttons : Byte;

begin
  InitMouse;
  Buttons:=DetectMouse;
  If Buttons=0 then
    WriteIn('No mouse present.')
  else
    WriteIn('Found mouse with ',Buttons,' buttons.');
  DoneMouse;
end.
```

66.5.2 DoneMouse

Synopsis: Deinitialize mouse driver.

Declaration: `procedure DoneMouse`

Visibility: default

Description: `DoneMouse` De-initializes the mouse driver. It cleans up any memory allocated when the mouse was initialized, or removes possible mouse hooks from memory. The mouse functions will not work after `DoneMouse` was called. If `DoneMouse` is called a second time, it will exit at once. `InitMouse` should be called before `DoneMouse` can be called again.

For an example, see most other mouse functions.

Errors: None.

See also: DetectMouse (1169), InitMouse (1173)

66.5.3 GetMouseButtons

Synopsis: Get the state of the mouse buttons.

Declaration: `function GetMouseButtons : Word`

Visibility: default

Description: `GetMouseButtons` returns the current button state of the mouse, i.e. it returns a or-ed combination of the following constants:

MouseLeftButtonWhen the left mouse button is held down.

MouseRightButtonWhen the right mouse button is held down.

MouseMiddleButtonWhen the middle mouse button is held down.

Errors: None.

See also: GetMouseEvent (1171), GetMouseX (1171), GetMouseY (1172)

Listing: ./examples/mouseex/ex2.pp

```
Program Example2;

{ Program to demonstrate the GetMouseButtons function. }

Uses mouse;

begin
  InitMouse;
  Writeln('Press right mouse button to exit program');
  While (GetMouseButtons<>MouseRightButton) do ;
  DoneMouse;
end.
```

66.5.4 GetMouseDriver

Synopsis: Get a copy of the currently active mouse driver.

Declaration: `procedure GetMouseDriver(var Driver: TMouseDriver)`

Visibility: default

Description: `GetMouseDriver` returns the currently set mouse driver. It can be used to retrieve the current mouse driver, and override certain callbacks.

A more detailed explanation about getting and setting mouse drivers can be found in mousedrv (1165).

For an example, see the section on writing a custom mouse driver, mousedrv (1165)

Errors: None.

See also: SetMouseDriver (1174)

66.5.5 GetMouseEvent

Synopsis: Get next mouse event from the queue.

Declaration: `procedure GetMouseEvent(var MouseEvent: TMouseEvent)`

Visibility: default

Description: `GetMouseEvent` returns the next mouse event (a movement, button press or button release), and waits for one if none is available in the queue.

Some mouse drivers can implement a mouse event queue which can hold multiple events till they are fetched. Others don't, and in that case, a one-event queue is implemented for use with PollMouseEvent (1173).

Errors: None.

See also: GetMouseButtons (1170), GetMouseX (1171), GetMouseY (1172)

66.5.6 GetMouseX

Synopsis: Query the current horizontal position of the mouse cursor.

Declaration: `function GetMouseX : Word`

Visibility: default

Description: `GetMouseX` returns the current `X` position of the mouse. `X` is measured in characters, starting at 0 for the left side of the screen.

Errors: None.

See also: GetMouseButtons (1170), GetMouseEvent (1171), GetMouseY (1172)

Listing: ./examples/mouseex/ex4.pp

```
Program Example4;

{ Program to demonstrate the GetMouseX,GetMouseY functions. }

Uses mouse;
```

```
Var
  X,Y : Word;

begin
  InitMouse;
  Writeln('Move mouse cursor to square 10,10 to end');
  Repeat
    X:=GetMouseX;
    Y:=GetMouseY;
    Writeln('X,Y= (',X,',',Y,')');
  Until (X=9) and (Y=9);
  DoneMouse;
end.
```

66.5.7 GetMouseY

Synopsis: Query the current vertical position of the mouse cursor.

Declaration: `function GetMouseY : Word`

Visibility: default

Description: `GetMouseY` returns the current `Y` position of the mouse. `Y` is measured in characters, starting at 0 for the top of the screen.

For an example, see GetMouseX (1171)

Errors: None.

See also: GetMouseButtons (1170), GetMouseEvent (1171), GetMouseX (1171)

66.5.8 HideMouse

Synopsis: Hide the mouse cursor.

Declaration: `procedure HideMouse`

Visibility: default

Description: `HideMouse` hides the mouse cursor. This may or may not be implemented on all systems, and depends on the driver.

Errors: None.

See also: ShowMouse (1175)

Listing: ./examples/mouseex/ex5.pp

```
Program Example5;

{ Program to demonstrate the HideMouse function. }

Uses mouse;

Var
  Event : TMouseEvent;
  Visible: Boolean;
```

```
begin
  InitMouse;
  ShowMouse;
  Visible:=True;
  Writeln('Press left mouse button to hide/show, right button quits');
  Repeat
   GetMouseEvent(Event);
   With Event do
     If (Buttons=MouseLeftbutton) and
        (Action=MouseActionDown) then
       begin
       If Visible then
         HideMouse
       else
         ShowMouse;
       Visible:=Not Visible;
       end;
  Until (Event.Buttons=MouseRightButton) and
        (Event.Action=MouseActionDown);
  DoneMouse;
end.
```

66.5.9 InitMouse

Synopsis: Initialize the FPC mouse driver.

Declaration: `procedure InitMouse`

Visibility: default

Description: `InitMouse` Initializes the mouse driver. This will allocate any data structures needed for the mouse to function. All mouse functions can be used after a call to `InitMouse`.

A call to `InitMouse` must always be followed by a call to DoneMouse (1170) at program exit. Failing to do so may leave the mouse in an unusable state, or may result in memory leaks.

For an example, see most other functions.

Errors: None.

See also: DoneMouse (1170), DetectMouse (1169)

66.5.10 PollMouseEvent

Synopsis: Query next mouse event. Do not wait if none available.

Declaration: `function PollMouseEvent(var MouseEvent: TMouseEvent) : Boolean`

Visibility: default

Description: `PollMouseEvent` checks whether a mouse event is available, and returns it in `MouseEvent` if one is found. The function result is `True` in that case. If no mouse event is pending, the function result is `False`, and the contents of `MouseEvent` is undefined.

Note that after a call to `PollMouseEvent`, the event should still be removed from the mouse event queue with a call to `GetMouseEvent`.

Errors: None.

See also: GetMouseEvent (1171), PutMouseEvent (1174)

66.5.11 PutMouseEvent

Synopsis: Put a mouse event in the event queue.

Declaration: `procedure PutMouseEvent(const MouseEvent: TMouseEvent)`

Visibility: default

Description: `PutMouseEvent` adds `MouseEvent` to the input queue. The next call to GetMouseEvent (1171) or `PollMouseEvent` will then return `MouseEvent`.

Please note that depending on the implementation the mouse event queue can hold only one value.

Errors: None.

See also: GetMouseEvent (1171), PollMouseEvent (1173)

66.5.12 SetMouseDriver

Synopsis: Set a new mouse driver.

Declaration: `procedure SetMouseDriver(const Driver: TMouseDriver)`

Visibility: default

Description: `SetMouseDriver` sets the mouse driver to `Driver`. This function should be called before InitMouse (1173) is called, or after `DoneMouse` is called. If it is called after the mouse has been initialized, it does nothing.

For more information on setting the mouse driver, mousedrv (1165).

For an example, see mousedrv (1165)

See also: InitMouse (1173), DoneMouse (1170), GetMouseDriver (1171)

66.5.13 SetMouseXY

Synopsis: Set the mouse cursor position.

Declaration: `procedure SetMouseXY(x: Word; y: Word)`

Visibility: default

Description: `SetMouseXY` places the mouse cursor on `X,Y`. X and Y are zero based character coordinates: `0,0` is the top-left corner of the screen, and the position is in character cells (i.e. not in pixels).

Errors: None.

See also: GetMouseX (1171), GetMouseY (1172)

Listing: ./examples/mouseex/ex7.pp

```
Program Example7;

{ Program to demonstrate the SetMouseXY function. }

Uses mouse;

begin
  InitMouse;
  Writeln('Click right mouse button to quit.');
```

```
  SetMouseXY(40,12);
  Repeat
    WriteIn(GetMouseX,',',GetMouseY);
    If (GetMouseX>70) then
      SetMouseXY(10,GetMouseY);
    If (GetMouseY>20) then
      SetMouseXY(GetMouseX,5);
  Until (GetMouseButtons=MouseRightButton);
  DoneMouse;
end.
```

66.5.14 ShowMouse

Synopsis: Show the mouse cursor.

Declaration: `procedure ShowMouse`

Visibility: default

Description: `ShowMouse` shows the mouse cursor if it was previously hidden. The capability to hide or show the mouse cursor depends on the driver.

For an example, see HideMouse (1172)

Errors: None.

See also: HideMouse (1172)

66.6 TMouseDriver

```
TMouseDriver = record
  UseDefaultQueue : Boolean;
  InitDriver :
  procedure;
  DoneDriver : procedure;
  DetectMouse : function : Byte
  ;
  ShowMouse : procedure;
  HideMouse : procedure;
  GetMouseX :
  function : Word;
  GetMouseY : function : Word;
  GetMouseButtons
  : function : Word;
  SetMouseXY : procedure(x: Word; y: Word);
  GetMouseEvent : procedure(var MouseEvent: TMouseEvent);
  PollMouseEvent
  : function(var MouseEvent: TMouseEvent) : Boolean;
  PutMouseEvent
  : procedure(const MouseEvent: TMouseEvent);
end
```

The `TMouseDriver` record is used to implement a mouse driver in the SetMouseDriver (1174) function. Its fields must be filled in before calling the SetMouseDriver (1174) function.

66.7 TMouseEvent

```
TMouseEvent = packed record
  buttons : Word;
  x : Word;
  y : Word
  ;
  Action : Word;
end
```

The `TMouseEvent` is the central type of the mouse unit, it is used to describe all mouse events.

The `Buttons` field describes which buttons were down when the event occurred. The `x,y` fields describe where the event occurred on the screen. The `Action` describes what action was going on when the event occurred. The `Buttons` and `Action` field can be examined using the constants defined in the unit interface.

Chapter 67

Reference for unit 'Objects'

67.1 Used units

Table 67.1: Used units by unit 'Objects'

Name	Page
System	1362

67.2 Overview

This document documents the objects unit. The unit was implemented by many people, and was mainly taken from the FreeVision sources. It has been ported to all supported platforms.

The methods and fields that are in a `Private` part of an object declaration have been left out of this documentation.

67.3 Constants, types and variables

67.3.1 Constants

```
coIndexError = - 1
```

Collection list error: Index out of range.

```
coOverflow = - 2
```

Collection list error: Overflow.

```
DefaultTPCompatible : Boolean = False
```

`DefaultTPCompatible` is used to initialize tstream.tpcompatible (**??**).

```
MaxBytes = 128 * 1024 * 128
```

Maximum data size (in bytes).

```
MaxCollectionSize = MaxBytes div SizeOf(Pointer)
```

Maximum collection size (in items).

```
MaxPtrs = MaxBytes div SizeOf(Pointer)
```

Maximum data size (in pointers).

```
MaxReadBytes = $7fffffff
```

Maximum data that can be read from a stream (not used).

```
MaxTPCompatibleCollectionSize = 65520 div 4
```

Maximum collection size (in items, same value as in TP).

```
MaxWords = MaxBytes div SizeOf(Word)
```

Maximum data size (in words).

```
RCollection : TStreamRec = (ObjType: 50; VmtLink: Ofs(^ TypeOf(TCollection
  )); Load: @ TCollection.Load; Store: @ TCollection.Store; Next: Nil
  )
```

Default stream record for the TCollection (1194) object.

```
RStrCollection : TStreamRec = (ObjType: 69; VmtLink: Ofs(^ TypeOf
  (TStrCollection)); Load: @ TStrCollection.Load; Store: @ TStrCollection
  .Store; Next: Nil)
```

Default stream record for the TStrCollection (1234) object.

```
RStringCollection : TStreamRec = (ObjType: 51; VmtLink: Ofs(^ TypeOf
  (TStringCollection)); Load: @ TStringCollection.Load; Store: @ TStringCollection
  .Store; Next: Nil)
```

Default stream record for the TStringCollection (1244) object.

```
RStringList : TStreamRec = (ObjType: 52; VmtLink: Ofs(^ TypeOf(TStringList
  )); Load: @ TStringList.Load; Store: Nil; Next: Nil)
```

Default stream record for the TStringList (1246) object.

```
RStrListMaker : TStreamRec = (ObjType: 52; VmtLink: Ofs(^ TypeOf(TStrListMaker
  )); Load: Nil; Store: @ TStrListMaker.Store; Next: Nil)
```

Default stream record for the TStrListMaker (1248) object.

```
stCreate = $3C00
```

Stream initialization mode: Create new file.

```
stError = - 1
```

Stream error codes: Access error.

```
stGetError = - 5
```

Stream error codes: Get object error.

```
stInitError = - 2
```

Stream error codes: Initialize error.

```
stOk = 0
```

Stream error codes: No error.

```
stOpen = $3D02
```

Stream initialization mode: Read/write access.

```
stOpenError = - 8
```

Stream error codes: Error opening stream.

```
stOpenRead = $3D00
```

Stream initialization mode: Read access only.

```
stOpenWrite = $3D01
```

Stream initialization mode: Write access only.

```
stPutError = - 6
```

Stream error codes: Put object error.

```
stReadError = - 3
```

Stream error codes: Stream read error.

```
StreamError : CodePointer = Nil
```

Pointer to default stream error handler.

```
stSeekError = - 7
```

Stream error codes: Seek error in stream.

```
stWriteError = - 4
```

Stream error codes: Stream write error.

```
vmtHeaderSize = 8
```

Size of the VMT header in an object (not used).

67.3.2 Types

```
AsciiZ = Array[0..255] of Char
```

Filename - null terminated array of characters.

```
FNameStr = String
```

Filename - shortstring version.

```
PBufStream = ^TBufStream
```

Pointer to TBufStream (1190) object.

```
PByteArray = ^TByteArray
```

Pointer to TByteArray (1182).

```
PCharSet = ^TCharSet
```

Pointer to TCharSet (1182).

```
PCollection = ^TCollection
```

Pointer to TCollection (1194) object.

```
PDosStream = ^TDosStream
```

Pointer to TDosStream (1209) object.

```
PItemList = ^TItemList
```

Pointer to TItemList (1182) object.

```
PMemoryStream = ^TMemoryStream
```

Pointer to TMemoryStream (1214) object.

```
PObject = ^TObject
```

Pointer to TObject (1216) object.

```
PPoint = ^TPoint
```

Pointer to TPoint (1218) record.

```
PPointerArray = ^TPointerArray
```

Pointer to TPointerArray (1182).

```
PRect = ^TRect
```

Pointer to TRect (1218) object.

```
PResourceCollection = ^TResourceCollection
```

Pointer to TResourceCollection (1224) object.

```
PResourceFile = ^TResourceFile
```

Pointer to TResourceFile (1225) object.

```
PSortedCollection = ^TSortedCollection
```

Pointer to TSortedCollection (1228) object.

```
PStrCollection = ^TStrCollection
```

Pointer to TStrCollection (1234) object.

```
PStream = ^TStream
```

Pointer type to TStream (1236).

```
PStreamRec = ^TStreamRec
```

Pointer to TStreamRec (1190).

```
PStrIndex = ^TStrIndex
```

Pointer to TStrIndex (1182) array.

```
PString = PShortString
```

Pointer to a shortstring.

```
PStringCollection = ^TStringCollection
```

Pointer to TStringCollection (1244) object.

```
PStringList = ^TStringList
```

Pointer to TStringList (1246) object.

```
PStrListMaker = ^TStrListMaker
```

Pointer to TStrListMaker (1248) object.

```
PUnSortedStrCollection = ^TUnSortedStrCollection
```

Pointer to TUnSortedStrCollection (1249) object.

```
PWordArray = ^TWordArray
```

Pointer to TWordArray (1182).

```
Sw_Integer = LongInt
```

Alias for longint.

```
Sw_Word = Cardinal
```

Alias for Cardinal.

```
TByteArray = Array[0..MaxBytes-1] of Byte
```

Array with maximum allowed number of bytes.

```
TCharSet = Set of Char
```

Generic set of characters type.

```
TItemList = Array[0..MaxCollectionSize-1] of Pointer
```

Pointer array type used in a TCollection (1194).

```
TPointerArray = Array[0..MaxPtrs-1] of Pointer
```

Array with maximum allowed number of pointers.

```
TStrIndex = Array[0..9999] of TStrIndexRec
```

Pointer array type used in a TStringList (1246).

```
TWordArray = Array[0..MaxWords-1] of Word
```

Array with maximum allowed number of words.

67.3.3 Variables

```
invalidhandle : THandle
```

Value for invalid handle. Initial value for file stream handles or when the stream is closed.

67.4 Procedures and functions

67.4.1 Abstract

Synopsis: Abstract error handler.

Declaration: `procedure Abstract`

Visibility: default

Description: When implementing abstract methods, do not declare them as `abstract`. Instead, define them simply as `virtual`. In the implementation of such abstract methods, call the `Abstract` procedure. This allows explicit control of what happens when an abstract method is called.

The current implementation of `Abstract` terminates the program with a run-time error 211.

Errors: None.

67.4.2 CallPointerConstructor

Synopsis: Call a constructor with a pointer argument.

Declaration:
```
function CallPointerConstructor(Ctor: CodePointer; Obj: pointer;
                                VMT: pointer; Param1: pointer) : pointer
```

Visibility: default

Description: `CallVoidConstructor` calls the constructor of an object. `Ctor` is the address of the constructor, `Obj` is a pointer to the instance. If it is `Nil`, then a new instance is allocated. `VMT` is a pointer to the object's VMT. `Param1` is passed to the constructor. The return value is a pointer to the instance.

Note that this can only be used on constructors that require a pointer as the sole argument. It can also be used to call a constructor with a single argument by reference.

Errors: If the constructor expects other arguments than a pointer, the stack may be corrupted.

See also: CallVoidConstructor (1184), CallPointerMethod (1183), CallVoidLocal (1184), CallPointerLocal (1183), CallVoidMethodLocal (1185), CallPointerMethodLocal (1184)

67.4.3 CallPointerLocal

Synopsis: Call a local nested function with a pointer argument.

Declaration:
```
function CallPointerLocal(Func: CodePointer; Frame: Pointer;
                          Param1: pointer) : pointer
```

Visibility: default

Description: `CallPointerLocal` calls the local procedure with address `Func`, where `Frame` is the frame of the wrapping function. It passes `Param1` to the local function.

Errors: If the local function expects other parameters than a pointer, the stack may become corrupted.

See also: CallPointerMethod (1183), CallVoidMethod (1185), CallVoidLocal (1184), CallVoidMethodLocal (1185), CallPointerMethodLocal (1184), CallVoidConstructor (1184), CallPointerConstructor (1183)

67.4.4 CallPointerMethod

Synopsis: Call a method with a single pointer argument.

Declaration:
```
function CallPointerMethod(Method: CodePointer; Obj: pointer;
                           Param1: pointer) : pointer
```

Visibility: default

Description: `CallPointerMethod` calls the method with address `Method` for instance `Obj`. It passes `Param1` to the method as the single argument. It returns a pointer to the instance.

Errors: If the method expects other parameters than a single pointer, the stack may become corrupted.

See also: CallVoidMethod (1185), CallVoidLocal (1184), CallPointerLocal (1183), CallVoidMethodLocal (1185), CallPointerMethodLocal (1184), CallVoidConstructor (1184), CallPointerConstructor (1183)

67.4.5 CallPointerMethodLocal

Synopsis: Call a local procedure of a method with a pointer argument.

Declaration:
```
function CallPointerMethodLocal(Func: CodePointer; Frame: Pointer;
                                Obj: pointer; Param1: pointer) : pointer
```

Visibility: default

Description: `CallPointerMethodLocal` calls the local procedure with address `Func`, where `Frame` is the frame of the wrapping method. It passes `Param1` to the local function.

Errors: If the local function expects other parameters than a pointer, the stack may become corrupted.

See also: CallPointerMethod (1183), CallVoidMethod (1185), CallPointerLocal (1183), CallVoidLocal (1184), CallVoidMethodLocal (1185), CallVoidConstructor (1184), CallPointerConstructor (1183)

67.4.6 CallVoidConstructor

Synopsis: Call a constructor with no arguments.

Declaration:
```
function CallVoidConstructor(Ctor: CodePointer; Obj: pointer;
                             VMT: pointer) : pointer
```

Visibility: default

Description: `CallVoidConstructor` calls the constructor of an object. `Ctor` is the address of the constructor, `Obj` is a pointer to the instance. If it is `Nil`, then a new instance is allocated. `VMT` is a pointer to the object's VMT. The return value is a pointer to the instance.

Note that this can only be used on constructors that require no arguments.

Errors: If the constructor expects arguments, the stack may be corrupted.

See also: CallPointerConstructor (1183), CallPointerMethod (1183), CallVoidLocal (1184), CallPointerLocal (1183), CallVoidMethodLocal (1185), CallPointerMethodLocal (1184)

67.4.7 CallVoidLocal

Synopsis: Call a local nested procedure.

Declaration:
```
function CallVoidLocal(Func: CodePointer; Frame: Pointer) : pointer
```

Visibility: default

Description: `CallVoidLocal` calls the local procedure with address `Func`, where `Frame` is the frame of the wrapping function.

Errors: If the local function expects parameters, the stack may become corrupted.

See also: CallPointerMethod (1183), CallVoidMethod (1185), CallPointerLocal (1183), CallVoidMethodLocal (1185), CallPointerMethodLocal (1184), CallVoidConstructor (1184), CallPointerConstructor (1183)

67.4.8 CallVoidMethod

Synopsis: Call an object method.

Declaration: `function CallVoidMethod(Method: CodePointer; Obj: pointer) : pointer`

Visibility: default

Description: `CallVoidMethod` calls the method with address `Method` for instance `Obj`. It returns a pointer to the instance.

Errors: If the method expects parameters, the stack may become corrupted.

See also: CallPointerMethod (1183), CallVoidLocal (1184), CallPointerLocal (1183), CallVoidMethodLocal (1185), CallPointerMethodLocal (1184), CallVoidConstructor (1184), CallPointerConstructor (1183)

67.4.9 CallVoidMethodLocal

Synopsis: Call a local procedure of a method.

Declaration:
```
function CallVoidMethodLocal(Func: CodePointer; Frame: Pointer;
                             Obj: pointer) : pointer
```

Visibility: default

Description: `CallVoidMethodLocal` calls the local procedure with address `Func`, where `Frame` is the frame of the wrapping method.

Errors: If the local function expects parameters, the stack may become corrupted.

See also: CallPointerMethod (1183), CallVoidMethod (1185), CallPointerLocal (1183), CallVoidLocal (1184), CallPointerMethodLocal (1184), CallVoidConstructor (1184), CallPointerConstructor (1183)

67.4.10 DisposeStr

Synopsis: Dispose of a shortstring which was allocated on the heap.

Declaration: `procedure DisposeStr(P: PString)`

Visibility: default

Description: `DisposeStr` removes a dynamically allocated string from the heap.

For an example, see NewStr (1186).

Errors: None.

See also: NewStr (1186), SetStr (1189)

67.4.11 LongDiv

Synopsis: Overflow safe divide.

Declaration: `function LongDiv(X: LongInt; Y: Integer) : Integer`

Visibility: default

Description: `LongDiv` divides `X` by `Y`. The result is of type `Integer` instead of type `Longint`, as you would get normally.

Errors: If Y is zero, a run-time error will be generated.

See also: LongMul (1186)

67.4.12 LongMul

Synopsis: Overflow safe multiply.

Declaration: `function LongMul(X: Integer; Y: Integer) : LongInt`

Visibility: default

Description: `LongMul` multiplies `X` with `Y`. The result is of type `Longint`. This avoids possible overflow errors you would normally get when multiplying `X` and `Y` that are too big.

Errors: None.

See also: LongDiv (1185)

67.4.13 NewStr

Synopsis: Allocate a copy of a shortstring on the heap.

Declaration: `function NewStr(const S: string) : PString`

Visibility: default

Description: `NewStr` makes a copy of the string `S` on the heap, and returns a pointer to this copy. If the string is empty then `Nil` is returned.

The allocated memory is not based on the declared size of the string passed to `NewStr`, but is based on the actual length of the string.

Errors: If not enough memory is available, an 'out of memory' error will occur.

See also: DisposeStr (1185), SetStr (1189)

Listing: ./examples/objectex/ex40.pp

```
Program ex40;

{ Program to demonstrate the NewStr function }

Uses Objects;

Var S : String;
    P : PString;

begin
  S:='Some really cute string';
  P:=NewStr(S);
  If P^<>S then
    WriteIn ('Oh-oh... Something is wrong !!');
  DisposeStr(P);
end.
```

67.4.14 RegisterObjects

Synopsis: Register standard objects.

Declaration: `procedure RegisterObjects`

Visibility: default

Description: `RegisterObjects` registers the following objects for streaming:

1. `TCollection`, see TCollection (1194).
2. `TStringCollection`, see TStringCollection (1244).
3. `TStrCollection`, see TStrCollection (1234).

Errors: None.

See also: RegisterType (1187)

67.4.15 RegisterType

Synopsis: Register new object for streaming.

Declaration: `procedure RegisterType(var S: TStreamRec)`

Visibility: default

Description: `RegisterType` registers a new type for streaming. An object cannot be streamed unless it has been registered first. The stream record `S` needs to have the following fields set:

ObjType: Sw_Word This should be a unique identifier. Each possible type should have it's own identifier.

VmtLink: pointer This should contain a pointer to the VMT (Virtual Method Table) of the object you try to register.

Load : Pointeris a pointer to a method that initializes an instance of that object, and reads the initial values from a stream. This method should accept as it's sole argument a `PStream` type variable.

Store: Pointeris a pointer to a method that stores an instance of the object to a stream. This method should accept as it's sole argument a `PStream` type variable.

The VMT of the object can be retrieved with the following expression:

```
VmtLink: Ofs(TypeOf(MyType)^);
```

Errors: In case of error (if a object with the same `ObjType`) is already registered), run-time error 212 occurs.

Listing: ./examples/objectex/myobject.pp

```
Unit MyObject;

Interface

Uses Objects;

Type
```

```
    PMyObject = ^TMyObject;
    TMyObject = Object(TObject)
      Field : Longint;
      Constructor Init;
      Constructor Load (Var Stream : TStream);
      Destructor Done;
      Procedure Store (Var Stream : TStream);
      Function  GetField : Longint;
      Procedure SetField (Value : Longint);
      end;

Implementation

Constructor TMyobject.Init;

begin
  Inherited Init;
  Field:=-1;
end;

Constructor TMyobject.Load (Var Stream : TStream);

begin
  Stream.Read(Field,Sizeof(Field));
end;

Destructor TMyObject.Done;

begin
end;

Function TMyObject.GetField : Longint;

begin
  GetField:=Field;
end;

Procedure TMyObject.SetField (Value : Longint);

begin
  Field:=Value;
end;

Procedure TMyObject.Store (Var Stream : TStream);

begin
  Stream.Write(Field,SizeOf(Field));
end;

Const MyObjectRec : TStreamRec = (
        Objtype : 666;
        vmtlink : Ofs(TypeOf(TMyObject)^);
        Load : @TMyObject.Load;
        Store : @TMyObject.Store;
        );

begin
  RegisterObjects;
```

```
  RegisterType (MyObjectRec);
end.
```

67.4.16 SetStr

Synopsis: Allocate a copy of a shortstring on the heap.

Declaration: `procedure SetStr(var p: PString; const s: string)`

Visibility: default

Description: `SetStr` makes a copy of the string `S` on the heap and returns the pointer to this copy in `P`. If `P` pointed to another string (i.e. was not `Nil`, the memory is released first. Contrary to NewStr (1186), if the string is empty then a pointer to an empty string is returned.

The allocated memory is not based on the declared size of the string passed to `NewStr`, but is based on the actual length of the string.

Errors: If not enough memory is available, an 'out of memory' error will occur.

See also: DisposeStr (1185), NewStr (1186)

67.5 LongRec

```
LongRec = packed record
  Hi : Word;
  Lo : Word;
end
```

Record describing a longint (in Words).

67.6 PtrRec

```
PtrRec = packed record
  Ofs : Word;
  Seg : Word;
end
```

Record describing a pointer to a memory location.

67.7 TStreamRec

```
TStreamRec = packed record
  ObjType : Sw_Word;
  VmtLink : pointer
  ;
  Load : CodePointer;
  Store : CodePointer;
```

```
  Next : PStreamRec
  ;
end
```

TSreamRec is used by the Objects unit streaming mechanism: when an object is registered, a TStreamRec record is added to a list of records. This list is used when objects need to be streamed from/streamed to a stream. It contains all the information needed to stream the object.

67.8 TStrIndexRec

```
TStrIndexRec = packed record
  Key : Sw_Word;
  Count : Word;
  Offset
  : Word;
end
```

Record type used in a TStringList (1246) to store the strings.

67.9 WordRec

```
WordRec = packed record
  Hi : Byte;
  Lo : Byte;
end
```

Record describing a Word (in bytes).

67.10 TBufStream

67.10.1 Description

Bufstream implements a buffered file stream. That is, all data written to the stream is written to memory first. Only when the buffer is full, or on explicit request, the data is written to disk.

Also, when reading from the stream, first the buffer is checked if there is any unread data in it. If so, this is read first. If not the buffer is filled again, and then the data is read from the buffer.

The size of the buffer is fixed and is set when constructing the file.

This is useful if you need heavy throughput for your stream, because it speeds up operations.

67.10.2 Method overview

Page	Method	Description
1192	Close	Flush data and Close the file.
1191	Done	Close the file and cleans up the instance.
1192	Flush	FLush data from buffer, and write it to stream.
1191	Init	Initialize an instance of `TBufStream` and open the file.
1193	Open	Open the file if it is closed.
1194	Read	Read data from the file to a buffer in memory.
1193	Seek	Set current position in file.
1193	Truncate	Flush buffer, and truncate the file at current position.
1194	Write	Write data to the file from a buffer in memory.

67.10.3 TBufStream.Init

Synopsis: Initialize an instance of `TBufStream` and open the file.

Declaration: `constructor Init(FileName: FNameStr; Mode: Word; Size: Word)`

Visibility: default

Description: `Init` instantiates an instance of `TBufStream`. The name of the file that contains (or will contain) the data of the stream is given in `FileName`. The `Mode` parameter determines whether a new file should be created and what access rights you have on the file. It can be one of the following constants:

stCreateCreates a new file.

stOpenReadRead access only.

stOpenWriteWrite access only.

stOpenRead and write access.

The `Size` parameter determines the size of the buffer that will be created. It should be different from zero.

For an example see TBufStream.Flush (1192).

Errors: On error, `Status` is set to `stInitError`, and `ErrorInfo` is set to the dos error code.

See also: TDosStream.Init (1210), TBufStream.Done (1191)

67.10.4 TBufStream.Done

Synopsis: Close the file and cleans up the instance.

Declaration: `destructor Done; Virtual`

Visibility: default

Description: `Done` flushes and closes the file if it was open and cleans up the instance of `TBufStream`.

For an example see TBufStream.Flush (1192).

Errors: None.

See also: TDosStream.Done (1210), TBufStream.Init (1191), TBufStream.Close (1192)

67.10.5 TBufStream.Close

Synopsis: Flush data and Close the file.

Declaration: `procedure Close; Virtual`

Visibility: default

Description: `Close` flushes and closes the file if it was open, and sets `Handle` to -1. Contrary to Done (1191) it does not clean up the instance of `TBufStream`

For an example see TBufStream.Flush (1192).

Errors: None.

See also: TStream.Close (1240), TBufStream.Init (1191), TBufStream.Done (1191)

67.10.6 TBufStream.Flush

Synopsis: FLush data from buffer, and write it to stream.

Declaration: `procedure Flush; Virtual`

Visibility: default

Description: When the stream is in write mode, the contents of the buffer are written to disk, and the buffer position is set to zero. When the stream is in read mode, the buffer position is set to zero.

Errors: Write errors may occur if the file was in write mode. see Write (1194) for more info on the errors.

See also: TStream.Close (1240), TBufStream.Init (1191), TBufStream.Done (1191)

Listing: ./examples/objectex/ex15.pp

```
Program ex15;

{ Program to demonstrate the TStream.Flush method }

Uses Objects;

Var L : String;
    P : PString;
    S : PBufStream; { Only one with Flush implemented. }

begin
  L:='Some constant string';
  { Buffer size of 100 }
  S:=New(PBufStream,Init('test.dat',stcreate,100));
  Writeln ('Writing "',L,'" to stream with handle ',S^.Handle);
  S^.WriteStr(@L);
  { At this moment, there is no data on disk yet. }
  S^.Flush;
  { Now there is. }
  S^.WriteStr(@L);
  { Close calls flush first }
  S^.Close;
  Writeln ('Closed stream. File handle is ',S^.Handle);
  S^.Open (stOpenRead);
  P:=S^.ReadStr;
  L:=P^;
```

```
  DisposeStr(P);
  WriteIn ('Read "',L,'" from stream with handle ',S^.Handle);
  S^.Close;
  Dispose (S,Done);
end.
```

67.10.7 TBufStream.Truncate

Synopsis: Flush buffer, and truncate the file at current position.

Declaration: `procedure Truncate;  Virtual`

Visibility: default

Description: If the status of the stream is `stOK`, then `Truncate` tries to flush the buffer, and then truncates the stream size to the current file position.

For an example, see TDosStream.Truncate (1211).

Errors: Errors can be those of Flush (1192) or TDosStream.Truncate (1211).

See also: TStream.Truncate (1241), TDosStream.Truncate (1211), TStream.GetSize (1238)

67.10.8 TBufStream.Seek

Synopsis: Set current position in file.

Declaration: `procedure Seek(Pos: LongInt);  Virtual`

Visibility: default

Description: If the stream's status is `stOK`, then `Seek` sets the file position to `Pos`. `Pos` is a zero-based offset, counted from the beginning of the file.

For an example, see TStream.Seek (1242);

Errors: In case an error occurs, the stream's status is set to `stSeekError`, and the OS error code is stored in `ErrorInfo`.

See also: TStream.Seek (1242), TStream.GetPos (1238)

67.10.9 TBufStream.Open

Synopsis: Open the file if it is closed.

Declaration: `procedure Open(OpenMode: Word);  Virtual`

Visibility: default

Description: If the stream's status is `stOK`, and the stream is closed then `Open` re-opens the file stream with mode `OpenMode`. This call can be used after a Close (1192) call.

For an example, see TDosStream.Open (1212).

Errors: If an error occurs when re-opening the file, then `Status` is set to `stOpenError`, and the OS error code is stored in `ErrorInfo`

See also: TStream.Open (1240), TBufStream.Close (1192)

67.10.10 TBufStream.Read

Synopsis: Read data from the file to a buffer in memory.

Declaration: `procedure Read(var Buf; Count: LongInt); Virtual`

Visibility: default

Description: If the Stream is open and the stream status is `stOK` then `Read` will read `Count` bytes from the stream and place them in `Buf`.

`Read` will first try to read the data from the stream's internal buffer. If insufficient data is available, the buffer will be filled before continuing to read. This process is repeated until all needed data has been read.

For an example, see TStream.Read (1243).

Errors: In case of an error, `Status` is set to `StReadError`, and `ErrorInfo` gets the OS specific error, or 0 when an attempt was made to read beyond the end of the stream.

See also: TStream.Read (1243), TBufStream.Write (1194)

67.10.11 TBufStream.Write

Synopsis: Write data to the file from a buffer in memory.

Declaration: `procedure Write(var Buf; Count: LongInt); Virtual`

Visibility: default

Description: If the Stream is open and the stream status is `stOK` then `Write` will write `Count` bytes from `Buf` and place them in the stream.

`Write` will first try to write the data to the stream's internal buffer. When the internal buffer is full, then the contents will be written to disk. This process is repeated until all data has been written.

For an example, see TStream.Read (1243).

Errors: In case of an error, `Status` is set to `StWriteError`, and `ErrorInfo` gets the OS specific error.

See also: TStream.Write (1243), TBufStream.Read (1194)

67.11 TCollection

67.11.1 Description

The `TCollection` object manages a collection of pointers or objects. It also provides a series of methods to manipulate these pointers or objects.

Whether or not objects are used depends on the kind of calls you use. All kinds come in 2 flavors, one for objects, one for pointers.

67.11.2 Method overview

Page	Method	Description
1196	At	Return the item at a certain index.
1205	AtDelete	Delete item at certain position.
1204	AtFree	Free an item at the indicates position, calling it's destructor.
1208	AtInsert	Insert an element at a certain position in the collection.
1207	AtPut	Set collection item, overwriting an existing value.
1203	Delete	Delete an item from the collection, but does not destroy it.
1202	DeleteAll	Delete all elements from the collection. Objects are not destroyed.
1196	Done	Clean up collection, release all memory.
1207	Error	Set error code.
1199	FirstThat	Return first item which matches a test.
1206	ForEach	Execute procedure for each item in the list.
1202	Free	Free item from collection, calling it's destructor.
1201	FreeAll	Release all objects from the collection.
1205	FreeItem	Destroy a non-nil item.
1198	GetItem	Read one item off the stream.
1197	IndexOf	Find the position of a certain item.
1195	Init	Instantiate a new collection.
1203	Insert	Insert a new item in the collection at the end.
1198	LastThat	Return last item which matches a test.
1195	Load	Initialize a new collection and load collection from a stream.
1200	Pack	Remove all `>Nil` pointers from the collection.
1209	PutItem	Put one item on the stream.
1207	SetLimit	Set maximum number of elements in the collection.
1209	Store	Write collection to a stream.

67.11.3 TCollection.Init

Synopsis: Instantiate a new collection.

Declaration: `constructor Init(ALimit: Sw_Integer; ADelta: Sw_Integer)`

Visibility: default

Description: `Init` initializes a new instance of a collection. It sets the (initial) maximum number of items in the collection to `ALimit`. `ADelta` is the increase size : The number of memory places that will be allocated in case `ALimit` is reached, and another element is added to the collection.

For an example, see TCollection.ForEach (1206).

Errors: None.

See also: TCollection.Load (1195), TCollection.Done (1196)

67.11.4 TCollection.Load

Synopsis: Initialize a new collection and load collection from a stream.

Declaration: `constructor Load(var S: TStream)`

Visibility: default

Description: `Load` initializes a new instance of a collection. It reads from stream `S` the item count, the item limit count, and the increase size. After that, it reads the specified number of items from the stream.

Errors: Errors returned can be those of GetItem (1198).

See also: TCollection.Init (1195), TCollection.GetItem (1198), TCollection.Done (1196)

Listing: ./examples/objectex/ex22.pp

```
Program ex22;

{ Program to demonstrate the TCollection.Load method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;
    S : PMemoryStream;

begin
  C:=New(PCollection,Init(100,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(100-I);
    C^.Insert(M);
    end;
  Writeln ('Inserted ',C^.Count,' objects');
  S:=New(PMemorySTream,Init(1000,10));
  C^.Store(S^);
  C^.FreeAll;
  // Dispose(C,Done);
  S^.Seek(0);
  C^.Load(S^);
  Writeln ('Read ',C^.Count,' objects from stream.');
  Dispose(S,Done);
  Dispose(C,Done);
end.
```

67.11.5 TCollection.Done

Synopsis: Clean up collection, release all memory.

Declaration: `destructor Done;  Virtual`

Visibility: default

Description: `Done` frees all objects in the collection, and then releases all memory occupied by the instance.

For an example, see TCollection.ForEach (1206).

Errors: None.

See also: TCollection.Init (1195), TCollection.FreeAll (1201)

67.11.6 TCollection.At

Synopsis: Return the item at a certain index.

Declaration: `function At(Index: Sw_Integer) : Pointer`

Visibility: default

Description: `At` returns the item at position `Index`.

Errors: If `Index` is less than zero or larger than the number of items in the collection, seepl{Error}{TCollection.Error} is called with `coIndexError` and `Index` as arguments, resulting in a run-time error.

See also: TCollection.Insert (1203)

Listing: ./examples/objectex/ex23.pp

```
Program ex23;

{ Program to demonstrate the TCollection.At method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

begin
  C:=New(PCollection,Init(100,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(100-I);
    C^.Insert(M);
    end;
  For I:=0 to C^.Count-1 do
    begin
    M:=C^.At(I);
    WriteIn ('Object ',i,' has field : ',M^.GetField);
    end;
  C^.FreeAll;
  Dispose(C,Done);
end.
```

67.11.7 TCollection.IndexOf

Synopsis: Find the position of a certain item.

Declaration: `function IndexOf(Item: Pointer) : Sw_Integer;  Virtual`

Visibility: default

Description: `IndexOf` returns the index of `Item` in the collection. If `Item` isn't present in the collection, -1 is returned.

Errors: If the item is not present, -1 is returned.

See also: TCollection.At (1196), TCollection.GetItem (1198), TCollection.Insert (1203)

Listing: ./examples/objectex/ex24.pp

```
Program ex24;

{ Program to demonstrate the TCollection.IndexOf method }
```

```
Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M,Keep : PMyObject;
    I : Longint;

begin
  Randomize;
  C:=New(PCollection,Init(100,10));
  Keep:=Nil;
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I-1);
    If Random<0.1 then
      Keep:=M;
    C^.Insert(M);
    end;
  If Keep=Nil then
    begin
    Writeln ('Please run again. No object selected');
    Halt(1);
    end;
  Writeln ('Selected object has field : ',Keep^.GetField);
  Write ('Selected object has index : ',C^.IndexOf(Keep));
  Writeln (' should match it''s field.');
  C^.FreeAll;
  Dispose(C,Done);
end.
```

67.11.8 TCollection.GetItem

Synopsis: Read one item off the stream.

Declaration: `function GetItem(var S: TStream) : Pointer;  Virtual`

Visibility: default

Description: `GetItem` reads a single item off the stream `S`, and returns a pointer to this item. This method is used internally by the Load method, and should not be used directly.

Errors: Possible errors are the ones from TStream.Get (1236).

See also: TStream.Get (1236), TCollection.Store (1209)

67.11.9 TCollection.LastThat

Synopsis: Return last item which matches a test.

Declaration: `function LastThat(Test: CodePointer) : Pointer`

Visibility: default

Description: This function returns the last item in the collection for which `Test` returns a non-nil result. `Test` is a function that accepts 1 argument: a pointer to an object, and that returns a pointer as a result.

Errors: None.

See also: TCollection.FirstThat (1199)

Listing: ./examples/objectex/ex25.pp

```
Program ex21;

{ Program to demonstrate the TCollection.Foreach method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

Function CheckField (Dummy: Pointer;P : PMyObject) : Longint;

begin
  If P^.GetField<56 then
    Checkfield:=1
  else
    CheckField:=0;
end;

begin
  C:=New(PCollection,Init(100,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I);
    C^.Insert(M);
    end;
  Writeln ('Inserted ',C^.Count,' objects');
  Writeln ('Last one for which Field<56  has index (should be 54) : ',
           C^.IndexOf(C^.LastThat(@CheckField)));
  C^.FreeAll;
  Dispose(C,Done);
end.
```

67.11.10 TCollection.FirstThat

Synopsis: Return first item which matches a test.

Declaration: `function FirstThat(Test: CodePointer) : Pointer`

Visibility: default

Description: This function returns the first item in the collection for which `Test` returns a non-nil result. `Test` is a function that accepts 1 argument: a pointer to an object, and that returns a pointer as a result.

Errors: None.

See also: TCollection.LastThat (1198)

Listing: ./examples/objectex/ex26.pp

```
Program ex21;

{ Program to demonstrate the TCollection.FirstThat method }
```

```
Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

Function CheckField (Dummy: Pointer;P : PMyObject) : Longint;

begin
  If P^.GetField>56 then
    Checkfield:=1
  else
    CheckField:=0;
end;

begin
  C:=New(PCollection,Init(100,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I);
    C^.Insert(M);
    end;
  Writeln ('Inserted ',C^.Count,' objects');
  Writeln ('first one for which Field>56  has index (should be 56) : ',
           C^.IndexOf(C^.FirstThat(@CheckField)));
  C^.FreeAll;
  Dispose(C,Done);
end.
```

67.11.11 TCollection.Pack

Synopsis: Remove all `>Nil` pointers from the collection.

Declaration: `procedure Pack`

Visibility: default

Description: `Pack` removes all `Nil` pointers from the collection, and adjusts `Count` to reflect this change. No memory is freed as a result of this call. In order to free any memory, you can call `SetLimit` with an argument of `Count` after a call to `Pack`.

Errors: None.

See also: TCollection.SetLimit (1207)

Listing: ./examples/objectex/ex26.pp

```
Program ex21;

{ Program to demonstrate the TCollection.FirstThat method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;
```

```
Function CheckField (Dummy: Pointer;P : PMyObject) : Longint;

begin
  If P^.GetField>56 then
    Checkfield:=1
  else
    CheckField:=0;
end;

begin
  C:=New(PCollection,Init(100,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I);
    C^.Insert(M);
    end;
  Writeln ('Inserted ',C^.Count,' objects');
  Writeln ('first one for which Field>56  has index (should be 56) : ',
           C^.IndexOf(C^.FirstThat(@CheckField)));
  C^.FreeAll;
  Dispose(C,Done);
end.
```

67.11.12 TCollection.FreeAll

Synopsis: Release all objects from the collection.

Declaration: `procedure FreeAll`

Visibility: default

Description: `FreeAll` calls the destructor of each object in the collection. It doesn't release any memory occupied by the collection itself, but it does set `Count` to zero.

See also: TCollection.DeleteAll (1202), TCollection.FreeItem (1205)

Listing: ./examples/objectex/ex28.pp

```
Program ex28;

{ Program to demonstrate the TCollection.FreeAll method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

begin
  Randomize;
  C:=New(PCollection,Init(120,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I-1);
    C^.Insert(M);
```

```
    end;
  Writeln ('Added 100 Items.');
  C^.FreeAll;
  Writeln ('Freed all objects.');
  Dispose(C,Done);
end.
```

67.11.13 TCollection.DeleteAll

Synopsis: Delete all elements from the collection. Objects are not destroyed.

Declaration: `procedure DeleteAll`

Visibility: default

Description: `DeleteAll` deletes all elements from the collection. It just sets the `Count` variable to zero. Contrary to FreeAll (1201), `DeletAll` doesn't call the destructor of the objects.

Errors: None.

See also: TCollection.FreeAll (1201), TCollection.Delete (1203)

Listing: ./examples/objectex/ex29.pp

```
Program ex29;

{
 Program to demonstrate the TCollection.DeleteAll method
 Compare with example 28, where FreeAll is used.
}

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

begin
  Randomize;
  C:=New(PCollection,Init(120,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I-1);
    C^.Insert(M);
    end;
  Writeln ('Added 100 Items.');
  C^.DeleteAll;
  Writeln ('Deleted all objects.');
  Dispose(C,Done);
end.
```

67.11.14 TCollection.Free

Synopsis: Free item from collection, calling it's destructor.

Declaration: `procedure Free(Item: Pointer)`

Visibility: default

Description: `Free` Deletes `Item` from the collection, and calls the destructor `Done` of the object.

Errors: If the `Item` is not in the collection, `Error` will be called with `coIndexError`.

See also: TCollection.FreeItem (1205)

Listing: ./examples/objectex/ex30.pp

```
Program ex30;

{ Program to demonstrate the TCollection.Free method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

begin
  Randomize;
  C:=New(PCollection,Init(120,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I-1);
    C^.Insert(M);
    end;
  Writeln ('Added 100 Items.');
  With C^ do
    While Count>0 do Free(At(Count-1));
  Writeln ('Freed all objects.');
  Dispose(C,Done);
end.
```

67.11.15 TCollection.Insert

Synopsis: Insert a new item in the collection at the end.

Declaration: `procedure Insert(Item: Pointer); Virtual`

Visibility: default

Description: `Insert` inserts `Item` in the collection. `TCollection` inserts this item at the end, but descendent objects may insert it at another place.

Errors: None.

See also: TCollection.AtInsert (1208), TCollection.AtPut (1207)

67.11.16 TCollection.Delete

Synopsis: Delete an item from the collection, but does not destroy it.

Declaration: `procedure Delete(Item: Pointer)`

Visibility: default

Description: `Delete` deletes `Item` from the collection. It doesn't call the item's destructor, though. For this the Free (1202) call is provided.

Errors: If the `Item` is not in the collection, `Error` will be called with `coIndexError`.

See also: TCollection.AtDelete (1205), TCollection.Free (1202)

Listing: ./examples/objectex/ex31.pp

```
Program ex31;

{ Program to demonstrate the TCollection.Delete method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

begin
  Randomize;
  C:=New(PCollection,Init(120,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I-1);
    C^.Insert(M);
    end;
  Writeln ('Added 100 Items.');
  With C^ do
    While Count>0 do Delete(At(Count-1));
  Writeln ('Freed all objects');
  Dispose(C,Done);
end.
```

67.11.17 TCollection.AtFree

Synopsis: Free an item at the indicates position, calling it's destructor.

Declaration: `procedure AtFree(Index: Sw_Integer)`

Visibility: default

Description: `AtFree` deletes the item at position `Index` in the collection, and calls the item's destructor if it is not `Nil`.

Errors: If `Index` isn't valid then Error (1207) is called with `CoIndexError`.

See also: TCollection.Free (1202), TCollection.AtDelete (1205)

Listing: ./examples/objectex/ex32.pp

```
Program ex32;

{ Program to demonstrate the TCollection.AtFree method }
```

```
Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

begin
  Randomize;
  C:=New(PCollection,Init(120,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I-1);
    C^.Insert(M);
    end;
  Writeln ('Added 100 Items');
  With C^ do
    While Count>0 do AtFree(Count-1);
  Writeln ('Freed all objects.');
  Dispose(C,Done);
end.
```

67.11.18 TCollection.FreeItem

Synopsis: Destroy a non-nil item.

Declaration: `procedure FreeItem(Item: Pointer);  Virtual`

Visibility: default

Description: `FreeItem` calls the destructor of `Item` if it is not nil.

Remark This function is used internally by the TCollection object, and should not be called directly.

Errors: None.

See also: TCollection.Free (1202), TCollection.AtFree (1204)

67.11.19 TCollection.AtDelete

Synopsis: Delete item at certain position.

Declaration: `procedure AtDelete(Index: Sw_Integer)`

Visibility: default

Description: `AtDelete` deletes the pointer at position `Index` in the collection. It doesn't call the object's destructor.

Errors: If `Index` isn't valid then Error (1207) is called with `CoIndexError`.

See also: TCollection.Delete (1203)

Listing: ./examples/objectex/ex33.pp

```
Program ex33;

{ Program to demonstrate the TCollection.AtDelete method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

begin
  Randomize;
  C:=New(PCollection,Init(120,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I-1);
    C^.Insert(M);
    end;
  Writeln ('Added 100 Items.');
  With C^ do
    While Count>0 do AtDelete(Count-1);
  Writeln ('Freed all objects.');
  Dispose(C,Done);
end.
```

67.11.20 TCollection.ForEach

Synopsis: Execute procedure for each item in the list.

Declaration: `procedure ForEach(Action: CodePointer)`

Visibility: default

Description: `ForEach` calls `Action` for each element in the collection, and passes the element as an argument to `Action`.

`Action` is a procedural type variable that accepts a pointer as an argument.

Errors: None.

See also: TCollection.FirstThat (1199), TCollection.LastThat (1198)

Listing: ./examples/objectex/ex21.pp

```
Program ex21;

{ Program to demonstrate the TCollection.Foreach method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

Procedure PrintField (Dummy: Pointer;P : PMyObject);

begin
```

```
  WriteLn ('Field : ',P^.GetField);
end;

begin
  C:=New(PCollection,Init(100,10));
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(100-I);
    C^.Insert(M);
    end;
  WriteLn ('Inserted ',C^.Count,' objects');
  C^.ForEach(@PrintField);
  C^.FreeAll;
  Dispose(C,Done);
end.
```

67.11.21 TCollection.SetLimit

Synopsis: Set maximum number of elements in the collection.

Declaration: `procedure SetLimit(ALimit: Sw_Integer);  Virtual`

Visibility: default

Description: `SetLimit` sets the maximum number of elements in the collection. `ALimit` must not be less than `Count`, and should not be larger than `MaxCollectionSize`

For an example, see Pack (1200).

Errors: None.

See also: TCollection.Init (1195)

67.11.22 TCollection.Error

Synopsis: Set error code.

Declaration: `procedure Error(Code: Integer; Info: Integer);  Virtual`

Visibility: default

Description: `Error` is called by the various `TCollection` methods in case of an error condition. The default behaviour is to make a call to `RunError` with an error of `212-Code`.

This method can be overridden by descendent objects to implement a different error-handling.

See also: Abstract (1182)

67.11.23 TCollection.AtPut

Synopsis: Set collection item, overwriting an existing value.

Declaration: `procedure AtPut(Index: Sw_Integer; Item: Pointer)`

Visibility: default

Description: `AtPut` sets the element at position `Index` in the collection to `Item`. Any previous value is overwritten.

For an example, see Pack (1200).

Errors: If `Index` isn't valid then Error (1207) is called with `CoIndexError`.

67.11.24 TCollection.AtInsert

Synopsis: Insert an element at a certain position in the collection.

Declaration: `procedure AtInsert(Index: Sw_Integer; Item: Pointer)`

Visibility: default

Description: `AtInsert` inserts `Item` in the collection at position `Index`, shifting all elements by one position. In case the current limit is reached, the collection will try to expand with a call to `SetLimit`

Errors: If `Index` isn't valid then Error (1207) is called with `CoIndexError`. If the collection fails to expand, then `coOverFlow` is passed to `Error`.

See also: TCollection.Insert (1203)

Listing: ./examples/objectex/ex34.pp

```
Program ex34;

{ Program to demonstrate the TCollection.AtInsert method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I : Longint;

Procedure PrintField (Dummy: Pointer;P : PMyObject);

begin
  Writeln ('Field : ',P^.GetField);
end;


begin
  Randomize;
  C:=New(PCollection,Init(120,10));
  Writeln ('Inserting 100 records at random places.');
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(I-1);
    If I=1 then
      C^.Insert(M)
    else
      With C^ do
        AtInsert(Random(Count),M);
    end;
  Writeln ('Values : ');
  C^.Foreach(@PrintField);
  Dispose(C,Done);
end.
```

67.11.25 TCollection.Store

Synopsis: Write collection to a stream.

Declaration: `procedure Store(var S: TStream)`

Visibility: default

Description: `Store` writes the collection to the stream `S`. It does this by writing the current `Count`, `Limit` and `Delta` to the stream, and then writing each item to the stream.

The contents of the stream are then suitable for instantiating another collection with Load (1195).

For an example, see TCollection.Load (1195).

Errors: Errors returned are those by TStream.Put (1241).

See also: TCollection.Load (1195), TCollection.PutItem (1209)

67.11.26 TCollection.PutItem

Synopsis: Put one item on the stream.

Declaration: `procedure PutItem(var S: TStream; Item: Pointer);  Virtual`

Visibility: default

Description: `PutItem` writes `Item` to stream `S`. This method is used internally by the `TCollection` object, and should not be called directly.

Errors: Errors are those returned by TStream.Put (1241).

See also: Store (1209), GetItem (1198)

67.12 TDosStream

67.12.1 Description

`TDosStream` is a stream that stores it's contents in a file. it overrides a couple of methods of TStream (1236) for this.

In addition to the fields inherited from `TStream` (see TStream (1236)), there are some extra fields, that describe the file. (mainly the name and the OS file handle)

No buffering in memory is done when using `TDosStream`. All data are written directly to the file. For a stream that buffers in memory, see TBufStream (1190).

67.12.2 Method overview

Page	Method	Description
1210	Close	Close the file.
1210	Done	Closes the file and cleans up the instance.
1210	Init	Instantiate a new instance of TDosStream.
1212	Open	Open the file stream.
1213	Read	Read data from the stream to a buffer.
1211	Seek	Set file position.
1211	Truncate	Truncate the file on the current position.
1213	Write	Write data from a buffer to the stream.

67.12.3 TDosStream.Init

Synopsis: Instantiate a new instance of TDosStream.

Declaration: `constructor Init(FileName: FNameStr; Mode: Word)`

Visibility: default

Description: `Init` instantiates an instance of `TDosStream`. The name of the file that contains (or will contain) the data of the stream is given in `FileName`. The `Mode` parameter determines whether a new file should be created and what access rights you have on the file. It can be one of the following constants:

stCreateCreates a new file.

stOpenReadRead access only.

stOpenWriteWrite access only.

stOpenRead and write access.

For an example, see TDosStream.Truncate (1211).

Errors: On error, Status (**??**) is set to `stInitError`, and `ErrorInfo` is set to the dos error code.

See also: TDosStream.Done (1210)

67.12.4 TDosStream.Done

Synopsis: Closes the file and cleans up the instance.

Declaration: `destructor Done;  Virtual`

Visibility: default

Description: `Done` closes the file if it was open and cleans up the instance of `TDosStream`.

for an example, see e.g. TDosStream.Truncate (1211).

Errors: None.

See also: TDosStream.Init (1210), TDosStream.Close (1210)

67.12.5 TDosStream.Close

Synopsis: Close the file.

Declaration: `procedure Close;  Virtual`

Visibility: default

Description: `Close` closes the file if it was open, and sets `Handle` to -1. Contrary to Done (1210) it does not clean up the instance of `TDosStream`

For an example, see TDosStream.Open (1212).

Errors: None.

See also: TStream.Close (1240), TDosStream.Init (1210), TDosStream.Done (1210)

67.12.6 TDosStream.Truncate

Synopsis: Truncate the file on the current position.

Declaration: `procedure Truncate; Virtual`

Visibility: default

Description: If the status of the stream is `stOK`, then `Truncate` tries to truncate the stream size to the current file position.

Errors: If an error occurs, the stream's status is set to `stError` and `ErrorInfo` is set to the OS error code.

See also: TStream.Truncate (1241), TStream.GetSize (1238)

Listing: ./examples/objectex/ex16.pp

```
Program ex16;

{ Program to demonstrate the TStream.Truncate method }

Uses Objects;

Var L : String;
    P : PString;
    S : PDosStream; { Only one with Truncate implemented. }

begin
  L:='Some constant string';
  { Buffer size of 100 }
  S:=New(PDosStream,Init('test.dat',stcreate));
  Writeln ('Writing "',L,'" to stream with handle ',S^.Handle);
  S^.WriteStr(@L);
  S^.WriteStr(@L);
  { Close calls flush first }
  S^.Close;
  S^.Open (stOpen);
  Writeln ('Size of stream is : ',S^.GetSize);
  P:=S^.ReadStr;
  L:=P^;
  DisposeStr(P);
  Writeln ('Read "',L,'" from stream with handle ',S^.Handle);
  S^.Truncate;
  Writeln ('Truncated stream. Size is : ',S^.GetSize);
  S^.Close;
  Dispose (S,Done);
end.
```

67.12.7 TDosStream.Seek

Synopsis: Set file position.

Declaration: `procedure Seek(Pos: LongInt); Virtual`

Visibility: default

Description: If the stream's status is `stOK`, then `Seek` sets the file position to `Pos`. `Pos` is a zero-based offset, counted from the beginning of the file.

Errors: In case an error occurs, the stream's status is set to `stSeekError`, and the OS error code is stored in `ErrorInfo`.

See also: TStream.Seek (1242), TStream.GetPos (1238)

Listing: ./examples/objectex/ex17.pp

```
Program ex17;

{ Program to demonstrate the TStream.Seek method }

Uses Objects;

Var L : String;
    Marker : Word;
    P : PString;
    S : PDosStream;

begin
  L:='Some constant string';
  { Buffer size of 100 }
  S:=New(PDosStream,Init('test.dat',stcreate));
  Writeln ('Writing "',L,'" to stream.');
  S^.WriteStr(@L);
  Marker:=S^.GetPos;
  Writeln ('Set marker at ',Marker);
  L:='Some other constant String';
  Writeln ('Writing "',L,'" to stream.');
  S^.WriteStr(@L);
  S^.Close;
  S^.Open (stOpenRead);
  Writeln ('Size of stream is : ',S^.GetSize);
  Writeln ('Seeking to marker');
  S^.Seek(Marker);
  P:=S^.ReadStr;
  L:=P^;
  DisposeStr(P);
  Writeln ('Read "',L,'" from stream.');
  S^.Close;
  Dispose (S,Done);
end.
```

67.12.8 TDosStream.Open

Synopsis: Open the file stream.

Declaration: `procedure Open(OpenMode: Word);  Virtual`

Visibility: default

Description: If the stream's status is `stOK`, and the stream is closed then `Open` re-opens the file stream with mode `OpenMode`. This call can be used after a Close (1210) call.

Errors: If an error occurs when re-opening the file, then `Status` is set to `stOpenError`, and the OS error code is stored in `ErrorInfo`

See also: TStream.Open (1240), TDosStream.Close (1210)

Listing: ./examples/objectex/ex14.pp

```
Program ex14;

{ Program to demonstrate the TStream.Close method }

Uses Objects;

Var L : String;
    P : PString;
    S : PDosStream; { Only one with Close implemented. }

begin
  L:='Some constant string';
  S:=New(PDosStream,Init('test.dat',stcreate));
  Writeln ('Writing "',L,'" to stream with handle ',S^.Handle);
  S^.WriteStr(@L);
  S^.Close;
  Writeln ('Closed stream. File handle is ',S^.Handle);
  S^.Open (stOpenRead);
  P:=S^.ReadStr;
  L:=P^;
  DisposeStr(P);
  Writeln ('Read "',L,'" from stream with handle ',S^.Handle);
  S^.Close;
  Dispose (S,Done);
end.
```

67.12.9 TDosStream.Read

Synopsis: Read data from the stream to a buffer.

Declaration: `procedure Read(var Buf; Count: LongInt);  Virtual`

Visibility: default

Description: If the Stream is open and the stream status is `stOK` then `Read` will read `Count` bytes from the stream and place them in `Buf`.

For an example, see TStream.Read (1243).

Errors: In case of an error, `Status` is set to `StReadError`, and `ErrorInfo` gets the OS specific error, or 0 when an attempt was made to read beyond the end of the stream.

See also: TStream.Read (1243), TDosStream.Write (1213)

67.12.10 TDosStream.Write

Synopsis: Write data from a buffer to the stream.

Declaration: `procedure Write(var Buf; Count: LongInt);  Virtual`

Visibility: default

Description: If the Stream is open and the stream status is `stOK` then `Write` will write `Count` bytes from `Buf` and place them in the stream.

For an example, see TStream.Read (1243).

Errors: In case of an error, `Status` is set to `StWriteError`, and `ErrorInfo` gets the OS specific error.

See also: TStream.Write (1243), TDosStream.Read (1213)

67.13 TMemoryStream

67.13.1 Description

The `TMemoryStream` object implements a stream that stores it's data in memory. The data is stored on the heap, with the possibility to specify the maximum amount of data, and the size of the memory blocks being used.

See also: TStream (1236)

67.13.2 Method overview

Page	Method	Description
1214	Done	Clean up memory and destroy the object instance.
1214	Init	Initialize memory stream, reserves memory for stream data.
1215	Read	Read data from the stream to a location in memory.
1215	Truncate	Set the stream size to the current position.
1216	Write	Write data to the stream.

67.13.3 TMemoryStream.Init

Synopsis: Initialize memory stream, reserves memory for stream data.

Declaration:
```
constructor Init(ALimit: LongInt; ABlockSize: Word)
```

Visibility: default

Description: `Init` instantiates a new `TMemoryStream` object. The memorystreamobject will initially allocate at least `ALimit` bytes memory, divided into memory blocks of size `ABlockSize`. The number of blocks needed to get to `ALimit` bytes is rounded up.

By default, the number of blocks is 1, and the size of a block is 8192. This is selected if you specify 0 as the blocksize.

For an example, see e.g TStream.CopyFrom (1244).

Errors: If the stream cannot allocate the initial memory needed for the memory blocks, then the stream's status is set to `stInitError`.

See also: TMemoryStream.Done (1214)

67.13.4 TMemoryStream.Done

Synopsis: Clean up memory and destroy the object instance.

Declaration:
```
destructor Done;  Virtual
```

Visibility: default

Description: `Done` releases the memory blocks used by the stream, and then cleans up the memory used by the stream object itself.

For an example, see e.g TStream.CopyFrom (1244).

Errors: None.

See also: TMemoryStream.Init (1214)

67.13.5 TMemoryStream.Truncate

Synopsis: Set the stream size to the current position.

Declaration: `procedure Truncate; Virtual`

Visibility: default

Description: `Truncate` sets the size of the memory stream equal to the current position. It de-allocates any memory-blocks that are no longer needed, so that the new size of the stream is the current position in the stream, rounded up to the first multiple of the stream blocksize.

Errors: If an error occurs during memory de-allocation, the stream's status is set to `stError`

See also: TStream.Truncate (1241)

Listing: ./examples/objectex/ex20.pp

```
Program ex20;

{ Program to demonstrate the TMemoryStream.Truncate method }

Uses Objects;

Var L : String;
    P : PString;
    S : PMemoryStream;
    I : Longint;

begin
  L:='Some constant string';
  { Buffer size of 100 }
  S:=New(PMemoryStream,Init(1000,100));
  Writeln ('Writing 100 times "',L,'" to stream.');
  For I:=1 to 100 do
    S^.WriteStr(@L);
  Writeln ('Finished.');
  S^.Seek(100);
  S^.Truncate;
  Writeln ('Truncated at byte 100.');
  Dispose (S,Done);
  Writeln ('Finished.');
end.
```

67.13.6 TMemoryStream.Read

Synopsis: Read data from the stream to a location in memory.

Declaration: `procedure Read(var Buf; Count: LongInt); Virtual`

Visibility: default

Description: `Read` reads `Count` bytes from the stream to `Buf`. It updates the position of the stream.

For an example, see TStream.Read (1243).

Errors: If there is not enough data available, no data is read, and the stream's status is set to `stReadError`.

See also: TStream.Read (1243), TMemoryStream.Write (1216)

67.13.7 TMemoryStream.Write

Synopsis: Write data to the stream.

Declaration: `procedure Write(var Buf; Count: LongInt);  Virtual`

Visibility: default

Description: `Write` copies `Count` bytes from `Buf` to the stream. It updates the position of the stream.

If not enough memory is available to hold the extra `Count` bytes, then the stream will try to expand, by allocating as much blocks with size `BlkSize` (as specified in the constructor call Init (1214)) as needed.

For an example, see TStream.Read (1243).

Errors: If the stream cannot allocate more memory, then the status is set to `stWriteError`

See also: TStream.Write (1243), TMemoryStream.Read (1215)

67.14 TObject

67.14.1 Description

This type serves as the basic object for all other objects in the Objects unit.

67.14.2 Method overview

Page	Method	Description
1217	Done	Destroy an object.
1217	Free	Destroy an object and release all memory.
1216	Init	Construct (initialize) a new object.
1217	Is_Object	Check whether a pointer points to an object.

67.14.3 TObject.Init

Synopsis: Construct (initialize) a new object.

Declaration: `constructor Init`

Visibility: default

Description: Instantiates a new object of type `TObject`. It fills the instance up with Zero bytes.

For an example, see Free (1217)

Errors: None.

See also: TObject.Free (1217), TObject.Done (1217)

67.14.4 TObject.Free

Synopsis: Destroy an object and release all memory.

Declaration: `procedure Free`

Visibility: default

Description: `Free` calls the destructor of the object, and releases the memory occupied by the instance of the object.

Errors: No checking is performed to see whether `self` is `nil` and whether the object is indeed allocated on the heap.

See also: TObject.Init (1216), TObject.Done (1217)

Listing: ./examples/objectex/ex7.pp

```
program ex7;

{ Program to demonstrate the TObject.Free call }

Uses Objects;

Var O : PObject;

begin
  // Allocate memory for object.
  O:=New(PObject,Init);
  // Free memory of object.
  O^.free;
end.
```

67.14.5 TObject.Is_Object

Synopsis: Check whether a pointer points to an object.

Declaration: `function Is_Object(P: Pointer) : Boolean`

Visibility: default

Description: `Is_Object` returns `True` if the pointer `P` points to an instance of a `TObject` descendent, it returns `false` otherwise.

67.14.6 TObject.Done

Synopsis: Destroy an object.

Declaration: `destructor Done; Virtual`

Visibility: default

Description: `Done`, the destructor of `TObject` does nothing. It is mainly intended to be used in the TObject.Free (1217) method.

The destructor Done does not free the memory occupied by the object.

Errors: None.

See also: TObject.Free (1217), TObject.Init (1216)

Listing: ./examples/objectex/ex8.pp

```
program ex8;

{ Program to demonstrate the TObject.Done call }

Uses Objects;

Var O : PObject;

begin
  // Allocate memory for object.
  O:=New(PObject,Init);
  O^.Done;
end.
```

67.15 TPoint

67.15.1 Description

Record describing a point in a 2 dimensional plane.

67.16 TRect

67.16.1 Description

Describes a rectangular region in a plane.

67.16.2 Method overview

Page	Method	Description
1223	Assign	Set rectangle corners.
1220	Contains	Determine if a point is inside the rectangle.
1220	Copy	Copy cornerpoints from another rectangle.
1218	Empty	Is the surface of the rectangle zero.
1219	Equals	Do the corners of the rectangles match.
1223	Grow	Expand rectangle with certain size.
1221	Intersect	Reduce rectangle to intersection with another rectangle.
1222	Move	Move rectangle along a vector.
1221	Union	Enlarges rectangle to encompass another rectangle.

67.16.3 TRect.Empty

Synopsis: Is the surface of the rectangle zero.

Declaration: `function Empty : Boolean`

Visibility: default

Description: `Empty` returns `True` if the rectangle defined by the corner points `A`, `B` has zero or negative surface.

Errors: None.

See also: TRect.Equals (1219), TRect.Contains (1220)

Listing: ./examples/objectex/ex1.pp

```
Program ex1;

{ Program to demonstrate TRect.Empty }

Uses objects;


Var ARect,BRect : TRect;
    P : TPoint;

begin
  With ARect.A do
    begin
    X:=10;
    Y:=10;
    end;
  With ARect.B do
    begin
    X:=20;
    Y:=20;
    end;
  { Offset B by (5,5) }
  With BRect.A do
    begin
    X:=15;
    Y:=15;
    end;
  With BRect.B do
    begin
    X:=25;
    Y:=25;
    end;
  { Point }
  With P do
    begin
    X:=15;
    Y:=15;
    end;
  Writeln ('A empty : ',ARect.Empty);
  Writeln ('B empty : ',BRect.Empty);
  Writeln ('A Equals B : ',ARect.Equals(BRect));
  Writeln ('A Contains (15,15) : ',ARect.Contains(P));
end.
```

67.16.4 TRect.Equals

Synopsis: Do the corners of the rectangles match.

Declaration: `function Equals(R: TRect) : Boolean`

Visibility: default

Description: `Equals` returns `True` if the rectangle has the same corner points `A,B` as the rectangle R, and `False` otherwise.

For an example, see TRect.Empty (1218)

Errors: None.

See also: TRect.Empty (1218), TRect.Contains (1220)

67.16.5 TRect.Contains

Synopsis: Determine if a point is inside the rectangle.

Declaration: `function Contains(P: TPoint) : Boolean`

Visibility: default

Description: `Contains` returns `True` if the point `P` is contained in the rectangle (including borders), `False` otherwise.

Errors: None.

See also: TRect.Intersect (1221), TRect.Equals (1219)

67.16.6 TRect.Copy

Synopsis: Copy cornerpoints from another rectangle.

Declaration: `procedure Copy(R: TRect)`

Visibility: default

Description: Assigns the rectangle R to the object. After the call to `Copy`, the rectangle R has been copied to the object that invoked `Copy`.

Errors: None.

See also: TRect.Assign (1223)

Listing: ./examples/objectex/ex2.pp

```
Program ex2;

{ Program to demonstrate TRect.Copy }

Uses objects;

Var ARect,BRect,CRect : TRect;

begin
  ARect.Assign(10,10,20,20);
  BRect.Assign(15,15,25,25);
  CRect.Copy(ARect);
  If ARect.Equals(CRect) Then
    WriteIn ('ARect equals CRect')
  Else
    WriteIn ('ARect does not equal CRect !');
end.
```

67.16.7 TRect.Union

Synopsis: Enlarges rectangle to encompass another rectangle.

Declaration: `procedure Union(R: TRect)`

Visibility: default

Description: `Union` enlarges the current rectangle so that it becomes the union of the current rectangle with the rectangle `R`.

Errors: None.

See also: TRect.Intersect (1221)

Listing: ./examples/objectex/ex3.pp

```
Program ex3;

{ Program to demonstrate TRect.Union }

Uses objects;


Var ARect,BRect,CRect : TRect;

begin
  ARect.Assign(10,10,20,20);
  BRect.Assign(15,15,25,25);
  { CRect is union of ARect and BRect }
  CRect.Assign(10,10,25,25);
  { Calculate it explicitly}
  ARect.Union(BRect);
  If ARect.Equals(CRect) Then
    WriteIn ('ARect equals CRect')
  Else
    WriteIn ('ARect does not equal CRect !');
end.
```

67.16.8 TRect.Intersect

Synopsis: Reduce rectangle to intersection with another rectangle.

Declaration: `procedure Intersect(R: TRect)`

Visibility: default

Description: `Intersect` makes the intersection of the current rectangle with `R`. If the intersection is empty, then the rectangle is set to the empty rectangle at coordinate (0,0).

Errors: None.

See also: TRect.Union (1221)

Listing: ./examples/objectex/ex4.pp

```
Program ex4;

{ Program to demonstrate TRect.Intersect }

Uses objects;


Var ARect,BRect,CRect : TRect;

begin
  ARect.Assign(10,10,20,20);
  BRect.Assign(15,15,25,25);
  { CRect is intersection of ARect and BRect }
  CRect.Assign(15,15,20,20);
  { Calculate it explicitly}
  ARect.Intersect(BRect);
  If ARect.Equals(CRect) Then
    WriteIn ('ARect equals CRect')
  Else
    WriteIn ('ARect does not equal CRect !');
  BRect.Assign(25,25,30,30);
  Arect.Intersect(BRect);
  If ARect.Empty Then
    WriteIn ('ARect is empty');
end.
```

67.16.9 TRect.Move

Synopsis: Move rectangle along a vector.

Declaration: `procedure Move(ADX: Sw_Integer; ADY: Sw_Integer)`

Visibility: default

Description: `Move` moves the current rectangle along a vector with components `(ADX,ADY)`. It adds `ADX` to the X-coordinate of both corner points, and `ADY` to both end points.

Errors: None.

See also: TRect.Grow (1223)

Listing: ./examples/objectex/ex5.pp

```
Program ex5;

{ Program to demonstrate TRect.Move }

Uses objects;


Var ARect,BRect : TRect;

begin
  ARect.Assign(10,10,20,20);
  ARect.Move(5,5);
  // Brect should be where new ARect is.
  BRect.Assign(15,15,25,25);
```

```
  If ARect.Equals(BRect) Then
    WriteIn ('ARect equals BRect')
  Else
    WriteIn ('ARect does not equal BRect !');
end.
```

67.16.10 TRect.Grow

Synopsis: Expand rectangle with certain size.

Declaration: `procedure Grow(ADX: Sw_Integer; ADY: Sw_Integer)`

Visibility: default

Description: `Grow` expands the rectangle with an amount `ADX` in the `X` direction (both on the left and right side of the rectangle, thus adding a length 2*ADX to the width of the rectangle), and an amount `ADY` in the `Y` direction (both on the top and the bottom side of the rectangle, adding a length 2*ADY to the height of the rectangle.

`ADX` and `ADY` can be negative. If the resulting rectangle is empty, it is set to the empty rectangle at `(0,0)`.

Errors: None.

See also: TRect.Move (1222)

Listing: ./examples/objectex/ex6.pp

```
Program ex6;

{ Program to demonstrate TRect.Grow }

Uses objects;


Var ARect,BRect : TRect;

begin
  ARect.Assign(10,10,20,20);
  ARect.Grow(5,5);
  // Brect should be where new ARect is.
  BRect.Assign(5,5,25,25);
  If ARect.Equals(BRect) Then
    WriteIn ('ARect equals BRect')
  Else
    WriteIn ('ARect does not equal BRect !');
end.
```

67.16.11 TRect.Assign

Synopsis: Set rectangle corners.

Declaration:
```
procedure Assign(XA: Sw_Integer; YA: Sw_Integer; XB: Sw_Integer;
                 YB: Sw_Integer)
```

Visibility: default

Description: `Assign` sets the corner points of the rectangle to `(XA,YA)` and `(Xb,Yb)`.

For an example, see TRect.Copy (1220).

Errors: None.

See also: TRect.Copy (1220)

67.17 TResourceCollection

67.17.1 Description

A `TResourceCollection` manages a collection of resource names. It stores the position and the size of a resource, as well as the name of the resource. It stores these items in records that look like this:

```
TYPE
   TResourceItem = packed RECORD
      Posn: LongInt;
      Size: LongInt;
      Key : String;
   End;
   PResourceItem = ^TResourceItem;
```

It overrides some methods of `TStringCollection` in order to accomplish this.

Remark Remark that the `TResourceCollection` manages the names of the resources and their associated positions and sizes, it doesn't manage the resources themselves.

67.17.2 Method overview

Page	Method	Description
1225	FreeItem	Release memory occupied by item.
1225	GetItem	Read an item from the stream.
1224	KeyOf	Return the key of an item in the collection.
1225	PutItem	Write an item to the stream.

67.17.3 TResourceCollection.KeyOf

Synopsis: Return the key of an item in the collection.

Declaration: `function KeyOf(Item: Pointer) : Pointer;  Virtual`

Visibility: default

Description: `KeyOf` returns the key of an item in the collection. For resources, the key is a pointer to the string with the resource name.

Errors: None.

See also: TStringCollection.Compare (1245)

67.17.4 TResourceCollection.GetItem

Synopsis: Read an item from the stream.

Declaration: `function GetItem(var S: TStream) : Pointer;  Virtual`

Visibility: default

Description: `GetItem` reads a resource item from the stream `S`. It reads the position, size and name from the stream, in that order. It DOES NOT read the resource itself from the stream.

The resulting item is not inserted in the collection. This call is manly for internal use by the TCollection.Load (1195) method.

Errors: Errors returned are those by TStream.Read (1243)

See also: TCollection.Load (1195), TStream.Read (1243)

67.17.5 TResourceCollection.FreeItem

Synopsis: Release memory occupied by item.

Declaration: `procedure FreeItem(Item: Pointer);  Virtual`

Visibility: default

Description: `FreeItem` releases the memory occupied by `Item`. It de-allocates the name, and then the resourceitem record.

It does NOT remove the item from the collection.

Errors: None.

See also: TCollection.FreeItem (1205)

67.17.6 TResourceCollection.PutItem

Synopsis: Write an item to the stream.

Declaration: `procedure PutItem(var S: TStream; Item: Pointer);  Virtual`

Visibility: default

Description: `PutItem` writes `Item` to the stream `S`. It does this by writing the position and size and name of the resource item to the stream.

This method is used primarily by the Store (1209) method.

Errors: Errors returned are those by TStream.Write (1243).

See also: TCollection.Store (1209)

67.18 TResourceFile

67.18.1 Description

TResourceFile (1225) represents the resources in a binary file image.

67.18.2 Method overview

Page	Method	Description
1226	Count	Number of resources in the file.
1228	Delete	Delete a resource from the file.
1226	Done	Destroy the instance and remove it from memory.
1227	Flush	Writes the resources to the stream.
1227	Get	Return a resource by key name.
1226	Init	Instantiate a new instance.
1227	KeyAt	Return the key of the item at a certain position.
1228	Put	Set a resource by key name.
1227	SwitchTo	Write resources to a new stream.

67.18.3 TResourceFile.Init

Synopsis: Instantiate a new instance.

Declaration: `constructor Init(AStream: PStream)`

Visibility: default

Description: `Init` instantiates a new instance of a `TResourceFile` object. If `AStream` is not nil then it is considered as a stream describing an executable image on disk.

`Init` will try to position the stream on the start of the resources section, and read all resources from the stream.

Errors: None.

See also: TResourceFile.Done (1226)

67.18.4 TResourceFile.Done

Synopsis: Destroy the instance and remove it from memory.

Declaration: `destructor Done;  Virtual`

Visibility: default

Description: `Done` cleans up the instance of the `TResourceFile` Object. If `Stream` was specified at initialization, then `Stream` is disposed of too.

Errors: None.

See also: TResourceFile.Init (1226)

67.18.5 TResourceFile.Count

Synopsis: Number of resources in the file.

Declaration: `function Count : Sw_Integer`

Visibility: default

Description: `Count` returns the number of resources. If no resources were read, zero is returned.

Errors: None.

See also: TResourceFile.Init (1226)

67.18.6 TResourceFile.KeyAt

Synopsis: Return the key of the item at a certain position.

Declaration: `function KeyAt(I: Sw_Integer) : string`

Visibility: default

Description: `KeyAt` returns the key (the name) of the `I`-th resource.

Errors: In case `I` is invalid, `TCollection.Error` will be executed.

See also: TResourceFile.Get (1227)

67.18.7 TResourceFile.Get

Synopsis: Return a resource by key name.

Declaration: `function Get(Key: string) : PObject`

Visibility: default

Description: `Get` returns a pointer to a instance of a resource identified by `Key`. If `Key` cannot be found in the list of resources, then `Nil` is returned.

Errors: Errors returned may be those by `TStream.Get`

67.18.8 TResourceFile.SwitchTo

Synopsis: Write resources to a new stream.

Declaration: `function SwitchTo(AStream: PStream; Pack: Boolean) : PStream`

Visibility: default

Description: `SwitchTo` switches to a new stream to hold the resources in. `AStream` will be the new stream after the call to `SwitchTo`.

If `Pack` is true, then all the known resources will be copied from the current stream to the new stream (`AStream`). If `Pack` is `False`, then only the current resource is copied.

The return value is the value of the original stream: `Stream`.

The `Modified` flag is set as a consequence of this call.

Errors: Errors returned can be those of TStream.Read (1243) and TStream.Write (1243).

See also: TResourceFile.Flush (1227)

67.18.9 TResourceFile.Flush

Synopsis: Writes the resources to the stream.

Declaration: `procedure Flush`

Visibility: default

Description: If the `Modified` flag is set to `True`, then `Flush` writes the resources to the stream `Stream`. It sets the `Modified` flag to true after that.

Errors: Errors can be those by TStream.Seek (1242) and TStream.Write (1243).

See also: TResourceFile.SwitchTo (1227)

67.18.10 TResourceFile.Delete

Synopsis: Delete a resource from the file.

Declaration: `procedure Delete(Key: string)`

Visibility: default

Description: `Delete` deletes the resource identified by `Key` from the collection. It sets the `Modified` flag to true.

Errors: None.

See also: TResourceFile.Flush (1227)

67.18.11 TResourceFile.Put

Synopsis: Set a resource by key name.

Declaration: `procedure Put(Item: PObject; Key: string)`

Visibility: default

Description: `Put` sets the resource identified by `Key` to `Item`. If no such resource exists, a new one is created. The item is written to the stream.

Errors: Errors returned may be those by TStream.Put (1241) and `TStream.Seek`

See also: Get (1227)

67.19 TSortedCollection

67.19.1 Description

`TSortedCollection` is an abstract class, implementing a sorted collection. You should never use an instance of `TSortedCollection` directly, instead you should declare a descendent type, and override the Compare (1230) method.

Because the collection is ordered, `TSortedCollection` overrides some `TCollection` methods, to provide faster routines for lookup.

The Compare (1230) method decides how elements in the collection should be ordered. Since `TCollection` has no way of knowing how to order pointers, you must override the compare method.

Additionally, `TCollection` provides a means to filter out duplicates. if you set `Duplicates` to `False` (the default) then duplicates will not be allowed.

The example below defines a descendent of `TSortedCollection` which is used in the examples.

67.19.2 Method overview

Page	Method	Description
1230	Compare	Compare two items in the collection.
1230	IndexOf	Return index of an item in the collection.
1229	Init	Instantiates a new instance of a `TSortedCollection`.
1232	Insert	Insert new item in collection.
1229	KeyOf	Return the key of an item.
1229	Load	Instantiates a new instance of a `TSortedCollection` and loads it from stream.
1231	Search	Search for item with given key.
1233	Store	Write the collection to the stream.

67.19.3 TSortedCollection.Init

Synopsis: Instantiates a new instance of a `TSortedCollection`.

Declaration: `constructor Init(ALimit: Sw_Integer; ADelta: Sw_Integer)`

Visibility: default

Description: `Init` calls the inherited constructor (see TCollection.Init (1195)) and sets the `Duplicates` flag to false.

You should not call this method directly, since `TSortedCollection` is a abstract class. Instead, the descendent classes should call it via the `inherited` keyword.

Errors: None.

See also: TSortedCollection.Load (1229), TCollection.Done (1196)

67.19.4 TSortedCollection.Load

Synopsis: Instantiates a new instance of a `TSortedCollection` and loads it from stream.

Declaration: `constructor Load(var S: TStream)`

Visibility: default

Description: `Load` calls the inherited constructor (see TCollection.Load (1195)) and reads the `Duplicates` flag from the stream..

You should not call this method directly, since `TSortedCollection` is a abstract class. Instead, the descendent classes should call it via the `inherited` keyword.

For an example, see TCollection.Load (1195).

Errors: None.

See also: TSortedCollection.Init (1229), TCollection.Done (1196)

67.19.5 TSortedCollection.KeyOf

Synopsis: Return the key of an item.

Declaration: `function KeyOf(Item: Pointer) : Pointer;  Virtual`

Visibility: default

Description: `KeyOf` returns the key associated with `Item`. `TSortedCollection` returns the item itself as the key, descendent objects can override this method to calculate a (unique) key based on the item passed (such as hash values).

`Keys` are used to sort the objects, they are used to search and sort the items in the collection. If descendent types override this method then it allows possibly for faster search/sort methods based on keys rather than on the objects themselves.

Errors: None.

See also: TSortedCollection.IndexOf (1230), TSortedCollection.Compare (1230)

67.19.6 TSortedCollection.IndexOf

Synopsis: Return index of an item in the collection.

Declaration: `function IndexOf(Item: Pointer) : Sw_Integer;  Virtual`

Visibility: default

Description: `IndexOf` returns the index of `Item` in the collection. It searches for the object based on it's key. If duplicates are allowed, then it returns the index of last object that matches `Item`.

In case `Item` is not found in the collection, -1 is returned.

For an example, see TCollection.IndexOf (1197)

Errors: None.

See also: TSortedCollection.Search (1231), TSortedCollection.Compare (1230)

67.19.7 TSortedCollection.Compare

Synopsis: Compare two items in the collection.

Declaration: `function Compare(Key1: Pointer; Key2: Pointer) : Sw_Integer;  Virtual`

Visibility: default

Description: `Compare` is an abstract method that should be overridden by descendent objects in order to compare two items in the collection. This method is used in the Search (1231) method and in the Insert (1232) method to determine the ordering of the objects.

The function should compare the two keys of items and return the following function results:

Result < 0 If `Key1` is logically before `Key2` (`Key1<Key2`)

Result = 0 If `Key1` and `Key2` are equal. (`Key1=Key2`)

Result > 0 If `Key1` is logically after `Key2` (`Key1>Key2`)

Errors: An 'abstract run-time error' will be generated if you call `TSortedCollection.Compare` directly.

See also: TSortedCollection.IndexOf (1230), TSortedCollection.Search (1231)

Listing: ./examples/objectex/mysortc.pp

```
Unit MySortC;

Interface

Uses Objects;

Type
  PMySortedCollection = ^TMySortedCollection;
  TMySortedCollection = Object(TSortedCollection)
      Function Compare (Key1,Key2 : Pointer): Sw_integer; virtual;
      end;

Implementation

Uses MyObject;

Function TMySortedCollection.Compare (Key1,Key2 : Pointer) :sw_integer;

begin
  Compare:=PMyobject(Key1)^.GetField - PMyObject(Key2)^.GetField;
end;

end.
```

67.19.8 TSortedCollection.Search

Synopsis: Search for item with given key.

Declaration: `function Search(Key: Pointer; var Index: Sw_Integer) : Boolean;  Virtual`

Visibility: default

Description: `Search` looks for the item with key `Key` and returns the position of the item (if present) in the collection in `Index`.

Instead of a linear search as `TCollection` does, `TSortedCollection` uses a binary search based on the keys of the objects. It uses the Compare (1230) function to implement this search.

If the item is found, `Search` returns `True`, otherwise `False` is returned.

Errors: None.

See also: TCollection.IndexOf (1197)

Listing: ./examples/objectex/ex36.pp

```
Program ex36;

{ Program to demonstrate the TSortedCollection.Insert method }

Uses Objects,MyObject,MySortC;
 { For TMyObject,TMySortedCollection definition and registration }

Var C : PSortedCollection;
    M : PMyObject;
    I : Longint;

Procedure PrintField (Dummy: Pointer;P : PMyObject);
```

```
begin
  WriteIn ('Field : ',P^.GetField);
end;


begin
  Randomize;
  C:=New(PMySortedCollection,Init(120,10));
  C^.Duplicates:=True;
  WriteIn ('Inserting 100 records at random places.');
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(Random(100));
    C^.Insert(M)
    end;
  M:=New(PMyObject,Init);
  Repeat;
    Write ('Value to search for (-1 stops) :');
    read (I);
    If I<>-1 then
      begin
      M^.SetField(i);
      If Not C^.Search (M,I) then
        WriteIn ('No such value found')
      else
        begin
        Write ('Value ',PMyObject(C^.At(I))^.GetField);
        WriteIn (' present at position ',I);
        end;
      end;
  Until I=-1;
  Dispose(M,Done);
  Dispose(C,Done);
end.
```

67.19.9 TSortedCollection.Insert

Synopsis: Insert new item in collection.

Declaration: `procedure Insert(Item: Pointer);  Virtual`

Visibility: default

Description: `Insert` inserts an item in the collection at the correct position, such that the collection is ordered at all times. You should never use Atinsert (1208), since then the collection ordering is not guaranteed.

If `Item` is already present in the collection, and `Duplicates` is `False`, the item will not be inserted.

Errors: None.

See also: TCollection.AtInsert (1208)

Listing: ./examples/objectex/ex35.pp

```
Program ex35;

{ Program to demonstrate the TSortedCollection.Insert method }

Uses Objects,MyObject,MySortC;
 { For TMyObject,TMySortedCollection definition and registration }

Var C : PSortedCollection;
    M : PMyObject;
    I : Longint;

Procedure PrintField (Dummy: Pointer;P : PMyObject);

begin
  Writeln ('Field : ',P^.GetField);
end;


begin
  Randomize;
  C:=New(PMySortedCollection,Init(120,10));
  Writeln ('Inserting 100 records at random places.');
  For I:=1 to 100 do
    begin
    M:=New(PMyObject,Init);
    M^.SetField(Random(100));
    C^.Insert(M)
    end;
  Writeln ('Values : ');
  C^.Foreach(@PrintField);
  Dispose(C,Done);
end.
```

67.19.10 TSortedCollection.Store

Synopsis: Write the collection to the stream.

Declaration: `procedure Store(var S: TStream)`

Visibility: default

Description: `Store` writes the collection to the stream `S`. It does this by calling the inherited TCollection.Store (1209), and then writing the `Duplicates` flag to the stream.

After a `Store`, the collection can be loaded from the stream with the constructor Load (1229)

For an example, see TCollection.Load (1195).

Errors: Errors can be those of TStream.Put (1241).

See also: TSortedCollection.Load (1229)

67.20 TStrCollection

67.20.1 Description

The `TStrCollection` object manages a sorted collection of null-terminated strings (pchar strings). To this end, it overrides the Compare (1230) method of `TSortedCollection`, and it introduces methods to read/write strings from a stream.

67.20.2 Method overview

Page	Method	Description
1234	Compare	Compare two strings in the collection.
1235	FreeItem	Free null-terminated string from the collection.
1235	GetItem	Read a null-terminated string from the stream.
1235	PutItem	Write a null-terminated string to the stream.

67.20.3 TStrCollection.Compare

Synopsis: Compare two strings in the collection.

Declaration: `function Compare(Key1: Pointer; Key2: Pointer) : Sw_Integer;  Virtual`

Visibility: default

Description: `TStrCollection` overrides the `Compare` function so it compares the two keys as if they were pointers to strings. The compare is done case sensitive. It returns

-1 if the first string is alphabetically earlier than the second string.

0 if the two strings are equal.

1 if the first string is alphabetically later than the second string.

Errors: None.

See also: TSortedCollection.Compare (1230)

Listing: ./examples/objectex/ex38.pp

```
Program ex38;

{ Program to demonstrate the TStrCollection.Compare method }

Uses Objects,Strings;

Var C : PStrCollection;
    S : String;
    I : longint;
    P : Pchar;

begin
  Randomize;
  C:=New(PStrCollection,Init(120,10));
  C^.Duplicates:=True; { Duplicates allowed }
  Writeln ('Inserting 100 records at random places.');
  For I:=1 to 100 do
    begin
    Str(Random(100),S);
```

```
    S:='String with value '+S;
    P:=StrAlloc(Length(S)+1);
    C^.Insert(StrPCopy(P,S));
    end;
  For I:=0 to 98 do
    With C^ do
      If Compare (At(I),At(I+1))=0 then
        WriteIn ('Duplicate string found at position ',I);
  Dispose(C,Done);
end.
```

67.20.4 TStrCollection.GetItem

Synopsis: Read a null-terminated string from the stream.

Declaration: `function GetItem(var S: TStream) : Pointer;  Virtual`

Visibility: default

Description: `GetItem` reads a null-terminated string from the stream `S` and returns a pointer to it. It doesn't insert the string in the collection.

This method is primarily introduced to be able to load and store the collection from and to a stream.

Errors: The errors returned are those of TStream.StrRead (1237).

See also: TStrCollection.PutItem (1235)

67.20.5 TStrCollection.FreeItem

Synopsis: Free null-terminated string from the collection.

Declaration: `procedure FreeItem(Item: Pointer);  Virtual`

Visibility: default

Description: `TStrCollection` overrides `FreeItem` so that the string pointed to by `Item` is disposed from memory.

Errors: None.

See also: TCollection.FreeItem (1205)

67.20.6 TStrCollection.PutItem

Synopsis: Write a null-terminated string to the stream.

Declaration: `procedure PutItem(var S: TStream; Item: Pointer);  Virtual`

Visibility: default

Description: `PutItem` writes the string pointed to by `Item` to the stream `S`.

This method is primarily used in the `Load` and `Store` methods, and should not be used directly.

Errors: Errors are those of TStream.StrWrite (1242).

See also: TStrCollection.GetItem (1235)

67.21 TStream

67.21.1 Description

The `TStream` object is the ancestor for all streaming objects, i.e. objects that have the capability to store and retrieve data.

It defines a number of methods that are common to all objects that implement streaming, many of them are virtual, and are only implemented in the descendent types.

Programs should not instantiate objects of type TStream directly, but instead instantiate a descendant type, such as `TDosStream`, `TMemoryStream`.

See also: PStream (1181), TDosStream (1209), TMemoryStream (1214)

67.21.2 Method overview

Page	Method	Description
1240	Close	Close the stream.
1244	CopyFrom	Copy data from another stream.
1242	Error	Set stream status.
1241	Flush	Flush the stream data from the buffer, if any.
1236	Get	Read an object definition from the stream.
1238	GetPos	Return current position in the stream.
1238	GetSize	Return the size of the stream.
1236	Init	Constructor for TStream instance.
1240	Open	Open the stream.
1241	Put	Write an object to the stream.
1243	Read	Read data from stream to buffer.
1239	ReadStr	Read a shortstring from the stream.
1240	Reset	Reset the stream.
1242	Seek	Set stream position.
1237	StrRead	Read a null-terminated string from the stream.
1242	StrWrite	Write a null-terminated string to the stream.
1241	Truncate	Truncate the stream size on current position.
1243	Write	Write a number of bytes to the stream.
1242	WriteStr	Write a pascal string to the stream.

67.21.3 TStream.Init

Synopsis: Constructor for TStream instance.

Declaration: `constructor Init`

Visibility: default

Description: `Init` initializes a TStream instance. Descendent streams should always call the inherited `Init`.

67.21.4 TStream.Get

Synopsis: Read an object definition from the stream.

Declaration: `function Get : PObject`

Visibility: default

Description: `Get` reads an object definition from a stream, and returns a pointer to an instance of this object.

Errors: On error, TStream.Status (**??**) is set, and `NIL` is returned.

See also: TStream.Put (1241)

Listing: ./examples/objectex/ex9.pp

```
Program ex9;

{ Program to demonstrate TStream.Get and TStream.Put }

Uses Objects,MyObject;  { Definition and registration of TMyObject}

Var Obj : PMyObject;
    S : PStream;

begin
  Obj:=New(PMyObject,Init);
  Obj^.SetField($1111) ;
  Writeln ('Field value : ',Obj^.GetField);
  { Since Stream is an abstract type, we instantiate a TMemoryStream }
  S:=New(PMemoryStream,Init(100,10));
  S^.Put(Obj);
  Writeln ('Disposing object');
  S^.Seek(0);
  Dispose(Obj,Done);
  Writeln ('Reading object');
  Obj:=PMyObject(S^.Get);
  Writeln ('Field Value : ',Obj^.GetField);
  Dispose(Obj,Done);
end.
```

67.21.5 TStream.StrRead

Synopsis: Read a null-terminated string from the stream.

Declaration: `function StrRead : PChar`

Visibility: default

Description: `StrRead` reads a string from the stream, allocates memory for it, and returns a pointer to a null-terminated copy of the string on the heap.

Errors: On error, `Nil` is returned.

See also: TStream.StrWrite (1242), TStream.ReadStr (1239)

Listing: ./examples/objectex/ex10.pp

```
Program ex10;

{
Program to demonstrate the TStream.StrRead TStream.StrWrite functions
}

Uses objects;
```

```
Var P : PChar;
    S : PStream;

begin
  P:='Constant Pchar string';
  Writeln ('Writing to stream : "',P,'"');
  S:=New(PMemoryStream,Init(100,10));
  S^.StrWrite(P);
  S^.Seek(0);
  P:=Nil;
  P:=S^.StrRead;
  DisPose (S,Done);
  Writeln ('Read from stream : "',P,'"');
  Freemem(P,Strlen(P)+1);
end.
```

67.21.6 TStream.GetPos

Synopsis: Return current position in the stream.

Declaration: `function GetPos : LongInt;  Virtual`

Visibility: default

Description: If the stream's status is `stOk`, `GetPos` returns the current position in the stream. Otherwise it returns `-1`

Errors: `-1` is returned if the status is an error condition.

See also: TStream.Seek (1242), TStream.GetSize (1238)

Listing: ./examples/objectex/ex11.pp

```
Program ex11;

{ Program to demonstrate the TStream.GetPos function }

Uses objects;

Var L : String;
    S : PStream;

begin
  L:='Some kind of string';
  S:=New(PMemoryStream,Init(100,10));
  Writeln ('Stream position before write: ',S^.GetPos);
  S^.WriteStr(@L);
  Writeln ('Stream position after write : ',S^.GetPos);
  Dispose(S,Done);
end.
```

67.21.7 TStream.GetSize

Synopsis: Return the size of the stream.

Declaration: `function GetSize : LongInt;  Virtual`

Visibility: default

Description: If the stream's status is `stOk` then `GetSize` returns the size of the stream, otherwise it returns `-1`.

Errors: `-1` is returned if the status is an error condition.

See also: TStream.Seek (1242), TStream.GetPos (1238)

Listing: ./examples/objectex/ex12.pp

```
Program ex12;

{ Program to demonstrate the TStream.GetSize function }

Uses objects;

Var L : String;
    S : PStream;

begin
  L:='Some kind of string';
  S:=New(PMemoryStream,Init(100,10));
  Writeln ('Stream size before write: ',S^.GetSize);
  S^.WriteStr(@L);
  Writeln ('Stream size after write : ',S^.GetSize);
  Dispose(S,Done);
end.
```

67.21.8 TStream.ReadStr

Synopsis: Read a shortstring from the stream.

Declaration: `function ReadStr : PString`

Visibility: default

Description: `ReadStr` reads a string from the stream, copies it to the heap and returns a pointer to this copy. The string is saved as a pascal string, and hence is NOT null terminated.

Errors: On error (e.g. not enough memory), `Nil` is returned.

See also: TStream.StrRead (1237)

Listing: ./examples/objectex/ex13.pp

```
Program ex13;

{
Program to demonstrate the TStream.ReadStr TStream.WriteStr functions
}

Uses objects;

Var P : PString;
    L : String;
    S : PStream;

begin
```

```
  L:='Constant string line';
  WriteLn ('Writing to stream : "',L,'"');
  S:=New(PMemoryStream,Init(100,10));
  S^.WriteStr(@L);
  S^.Seek(0);
  P:=S^.ReadStr;
  L:=P^;
  DisposeStr(P);
  DisPose (S,Done);
  WriteLn ('Read from stream : "',L,'"');
end.
```

67.21.9 TStream.Open

Synopsis: Open the stream.

Declaration: `procedure Open(OpenMode: Word);  Virtual`

Visibility: default

Description: `Open` is an abstract method, that should be overridden by descendent objects. Since opening a stream depends on the stream's type this is not surprising.

For an example, see TDosStream.Open (1212).

Errors: None.

See also: TStream.Close (1240), TStream.Reset (1240)

67.21.10 TStream.Close

Synopsis: Close the stream.

Declaration: `procedure Close;  Virtual`

Visibility: default

Description: `Close` is an abstract method, that should be overridden by descendent objects. Since Closing a stream depends on the stream's type this is not surprising.

for an example, see TDosStream.Open (1212).

Errors: None.

See also: TStream.Open (1240), TStream.Reset (1240)

67.21.11 TStream.Reset

Synopsis: Reset the stream.

Declaration: `procedure Reset`

Visibility: default

Description: `Reset` sets the stream's status to `0`, as well as the ErrorInfo

Errors: None.

See also: TStream.Open (1240), TStream.Close (1240)

67.21.12 TStream.Flush

Synopsis: Flush the stream data from the buffer, if any.

Declaration: `procedure Flush; Virtual`

Visibility: default

Description: `Flush` is an abstract method that should be overridden by descendent objects. It serves to enable the programmer to tell streams that implement a buffer to clear the buffer.

for an example, see TBufStream.Flush (1192).

Errors: None.

See also: TStream.Truncate (1241)

67.21.13 TStream.Truncate

Synopsis: Truncate the stream size on current position.

Declaration: `procedure Truncate; Virtual`

Visibility: default

Description: `Truncate` is an abstract procedure that should be overridden by descendent objects. It serves to enable the programmer to truncate the size of the stream to the current file position.

For an example, see TDosStream.Truncate (1211).

Errors: None.

See also: TStream.Seek (1242)

67.21.14 TStream.Put

Synopsis: Write an object to the stream.

Declaration: `procedure Put(P: PObject)`

Visibility: default

Description: `Put` writes the object pointed to by `P.P` should be non-nil. The object type must have been registered with RegisterType (1187).

After the object has been written, it can be read again with Get (1236).

For an example, see TStream.Get (1236);

Errors: No check is done whether P is `Nil` or not. Passing `Nil` will cause a run-time error 216 to be generated. If the object has not been registered, the status of the stream will be set to `stPutError`.

See also: TStream.Get (1236)

67.21.15 TStream.StrWrite

Synopsis: Write a null-terminated string to the stream.

Declaration: `procedure StrWrite(P: PChar)`

Visibility: default

Description: `StrWrite` writes the null-terminated string `P` to the stream. `P` can only be 65355 bytes long.

For an example, see TStream.StrRead (1237).

Errors: None.

See also: TStream.WriteStr (1242), TStream.StrRead (1237), TStream.ReadStr (1239)

67.21.16 TStream.WriteStr

Synopsis: Write a pascal string to the stream.

Declaration: `procedure WriteStr(P: PString)`

Visibility: default

Description: `StrWrite` writes the pascal string pointed to by `P` to the stream.

For an example, see TStream.ReadStr (1239).

Errors: None.

See also: TStream.StrWrite (1242), TStream.StrRead (1237), TStream.ReadStr (1239)

67.21.17 TStream.Seek

Synopsis: Set stream position.

Declaration: `procedure Seek(Pos: LongInt);  Virtual`

Visibility: default

Description: Seek sets the position to `Pos`. This position is counted from the beginning, and is zero based. (i.e. seek(0) sets the position pointer on the first byte of the stream)

For an example, see TDosStream.Seek (1211).

Errors: If `Pos` is larger than the stream size, `Status` is set to `StSeekError`.

See also: TStream.GetPos (1238), TStream.GetSize (1238)

67.21.18 TStream.Error

Synopsis: Set stream status.

Declaration: `procedure Error(Code: Integer; Info: Integer);  Virtual`

Visibility: default

Description: `Error` sets the stream's status to `Code` and `ErrorInfo` to `Info`. If the `StreamError` procedural variable is set, `Error` executes it, passing `Self` as an argument.

This method should not be called directly from a program. It is intended to be used in descendent objects.

Errors: None.

67.21.19 TStream.Read

Synopsis: Read data from stream to buffer.

Declaration: `procedure Read(var Buf; Count: LongInt); Virtual`

Visibility: default

Description: `Read` is an abstract method that should be overridden by descendent objects.

`Read` reads `Count` bytes from the stream into `Buf`. It updates the position pointer, increasing it's value with `Count`. `Buf` must be large enough to contain `Count` bytes.

Errors: No checking is done to see if `Buf` is large enough to contain `Count` bytes.

See also: TStream.Write (1243), TStream.ReadStr (1239), TStream.StrRead (1237)

Listing: ./examples/objectex/ex18.pp

```
program ex18;

{ Program to demonstrate the TStream.Read method }

Uses Objects;

Var Buf1,Buf2 : Array[1..1000] of Byte;
    I : longint;
    S : PMemorySTream;

begin
  For I:=1 to 1000 do
    Buf1[I]:=Random(1000);
  Buf2:=Buf1;
  S:=New(PMemoryStream,Init(100,10));
  S^.Write(Buf1,SizeOf(Buf1));
  S^.Seek(0);
  For I:=1 to 1000 do
    Buf1[I]:=0;
  S^.Read(Buf1,SizeOf(Buf1));
  For I:=1 to 1000 do
    If Buf1[I]<>buf2[i] then
      WriteIn ('Buffer differs at position ',I);
  Dispose(S,Done);
end.
```

67.21.20 TStream.Write

Synopsis: Write a number of bytes to the stream.

Declaration: `procedure Write(var Buf; Count: LongInt); Virtual`

Visibility: default

Description: `Write` is an abstract method that should be overridden by descendent objects.

`Write` writes `Count` bytes to the stream from `Buf`. It updates the position pointer, increasing it's value with `Count`.

For an example, see TStream.Read (1243).

Errors: No checking is done to see if `Buf` actually contains `Count` bytes.

See also: TStream.Read (1243), TStream.WriteStr (1242), TStream.StrWrite (1242)

67.21.21 TStream.CopyFrom

Synopsis: Copy data from another stream.

Declaration: `procedure CopyFrom(var S: TStream; Count: LongInt)`

Visibility: default

Description: `CopyFrom` reads Count bytes from stream `S` and stores them in the current stream. It uses the Read (1243) method to read the data, and the Write (1243) method to write in the current stream.

Errors: None.

See also: Read (1243), Write (1243)

Listing: ./examples/objectex/ex19.pp

```
Program ex19;

{ Program to demonstrate the TStream.CopyFrom function }

Uses objects;

Var P : PString;
    L : String;
    S1,S2 : PStream;

begin
  L:='Constant string line';
  Writeln ('Writing to stream 1 : "',L,'"');
  S1:=New(PMemoryStream,Init(100,10));
  S2:=New(PMemoryStream,Init(100,10));
  S1^.WriteStr(@L);
  S1^.Seek(0);
  Writeln ('Copying contents of stream 1 to stream 2');
  S2^.Copyfrom(S1^,S1^.GetSize);
  S2^.Seek(0);
  P:=S2^.ReadStr;
  L:=P^;
  DisposeStr(P);
  Dispose (S1,Done);
  Dispose (S2,Done);
  Writeln ('Read from stream 2 : "',L,'"');
end.
```

67.22 TStringCollection

67.22.1 Description

The `TStringCollection` object manages a sorted collection of pascal strings. To this end, it overrides the Compare (1230) method of `TSortedCollection`, and it introduces methods to read/write strings from a stream.

67.22.2 Method overview

Page	Method	Description
1245	Compare	Compare two strings in the collection.
1246	FreeItem	Dispose a string in the collection from memory.
1245	GetItem	Get string from the stream.
1246	PutItem	Write a string to the stream.

67.22.3 TStringCollection.GetItem

Synopsis: Get string from the stream.

Declaration: `function GetItem(var S: TStream) : Pointer;  Virtual`

Visibility: default

Description: `GetItem` reads a string from the stream `S` and returns a pointer to it. It doesn't insert the string in the collection.

This method is primarily introduced to be able to load and store the collection from and to a stream.

Errors: The errors returned are those of TStream.ReadStr (1239).

See also: TStringCollection.PutItem (1246)

67.22.4 TStringCollection.Compare

Synopsis: Compare two strings in the collection.

Declaration: `function Compare(Key1: Pointer; Key2: Pointer) : Sw_Integer;  Virtual`

Visibility: default

Description: `TStringCollection` overrides the `Compare` function so it compares the two keys as if they were pointers to strings. The compare is done case sensitive. It returns the following results:

-1 if the first string is alphabetically earlier than the second string.

0 if the two strings are equal.

1 if the first string is alphabetically later than the second string.

Errors: None.

See also: TSortedCollection.Compare (1230)

Listing: ./examples/objectex/ex37.pp

```
Program ex37;

{ Program to demonstrate the TStringCollection.Compare method }

Uses Objects;

Var C : PStringCollection;
    S : String;
    I : longint;

begin
  Randomize;
```

```
  C:=New(PStringCollection,Init(120,10));
  C^.Duplicates:=True; { Duplicates allowed }
  WriteLn ('Inserting 100 records at random places.');
  For I:=1 to 100 do
    begin
    Str(Random(100),S);
    S:='String with value '+S;
    C^.Insert(NewStr(S));
    end;
  For I:=0 to 98 do
    With C^ do
    If Compare (At(i),At(I+1))=0 then
      WriteLn ('Duplicate string found at position ',i);
  Dispose(C,Done);
end.
```

67.22.5 TStringCollection.FreeItem

Synopsis: Dispose a string in the collection from memory.

Declaration: `procedure FreeItem(Item: Pointer);  Virtual`

Visibility: default

Description: `TStringCollection` overrides `FreeItem` so that the string pointed to by `Item` is disposed from memory.

Errors: None.

See also: TCollection.FreeItem (1205)

67.22.6 TStringCollection.PutItem

Synopsis: Write a string to the stream.

Declaration: `procedure PutItem(var S: TStream; Item: Pointer);  Virtual`

Visibility: default

Description: `PutItem` writes the string pointed to by `Item` to the stream `S`.

This method is primarily used in the `Load` and `Store` methods, and should not be used directly.

Errors: Errors are those of TStream.WriteStr (1242).

See also: TStringCollection.GetItem (1245)

67.23 TStringList

67.23.1 Description

A `TStringList` object can be used to read a collection of strings stored in a stream. If you register this object with the RegisterType (1187) function, you cannot register the `TStrListMaker` object.

67.23.2 Method overview

Page	Method	Description
1247	Done	Clean up the instance.
1247	Get	Return a string by key name.
1247	Load	Load stringlist from stream.

67.23.3 TStringList.Load

Synopsis: Load stringlist from stream.

Declaration: `constructor Load(var S: TStream)`

Visibility: default

Description: The `Load` constructor reads the `TStringList` object from the stream `S`. It also reads the descriptions of the strings from the stream. The string descriptions are stored as an array of `TstrIndexrec` records, where each record describes a string on the stream. These records are kept in memory.

Errors: If an error occurs, a stream error is triggered.

See also: TStringList.Done (1247)

67.23.4 TStringList.Done

Synopsis: Clean up the instance.

Declaration: `destructor Done; Virtual`

Visibility: default

Description: The `Done` destructor frees the memory occupied by the string descriptions, and destroys the object.

Errors: None.

See also: Load (1247), TObject.Done (1217)

67.23.5 TStringList.Get

Synopsis: Return a string by key name.

Declaration: `function Get(Key: Sw_Word) : string`

Visibility: default

Description: `Get` reads the string with key `Key` from the list of strings on the stream, and returns this string. If there is no string with such a key, an empty string is returned.

Errors: If no string with key `Key` is found, an empty string is returned. A stream error may result if the stream doesn't contain the needed strings.

See also: TStrListMaker.Put (1248)

67.24 TStrListMaker

67.24.1 Description

The `TStrListMaker` object can be used to generate a stream with strings, which can be read with the `TStringList` object. If you register this object with the RegisterType (1187) function, you cannot register the `TStringList` object.

67.24.2 Method overview

Page	Method	Description
1248	Done	Clean up the instance and free all related memory.
1248	Init	Instantiate a new instance of `TStrListMaker`.
1248	Put	Add a new string to the list with associated key.
1249	Store	Write the strings to the stream.

67.24.3 TStrListMaker.Init

Synopsis: Instantiate a new instance of `TStrListMaker`.

Declaration: `constructor Init(AStrSize: Sw_Word; AIndexSize: Sw_Word)`

Visibility: default

Description: The `Init` constructor creates a new instance of the `TstrListMaker` object. It allocates `AStrSize` bytes on the heap to hold all the strings you wish to store. It also allocates enough room for `AIndexSize` key description entries (of the type `TStrIndexrec`).

`AStrSize` must be large enough to contain all the strings you wish to store. If not enough memory is allocated, other memory will be overwritten. The same is true for `AIndexSize` : maximally `AIndexSize` strings can be written to the stream.

Errors: None.

See also: TObject.Init (1216), TStrListMaker.Done (1248)

67.24.4 TStrListMaker.Done

Synopsis: Clean up the instance and free all related memory.

Declaration: `destructor Done;  Virtual`

Visibility: default

Description: The `Done` destructor de-allocates the memory for the index description records and the string data, and then destroys the object.

Errors: None.

See also: TObject.Done (1217), TStrListMaker.Init (1248)

67.24.5 TStrListMaker.Put

Synopsis: Add a new string to the list with associated key.

Declaration: `procedure Put(Key: Sw_Word; S: string)`

Visibility: default

Description: `Put` adds they string `S` with key `Key` to the collection of strings. This action doesn't write the string to a stream. To write the strings to the stream, see the Store (1249) method.

Errors: None.

See also: TStrListMaker.Store (1249)

67.24.6 TStrListMaker.Store

Synopsis: Write the strings to the stream.

Declaration: `procedure Store(var S: TStream)`

Visibility: default

Description: `Store` writes the collection of strings to the stream `S`. The collection can then be read with the `TStringList` object.

Errors: A stream error may occur when writing the strings to the stream.

See also: TStringList.Load (1247), TStrListMaker.Put (1248)

67.25 TUnSortedStrCollection

67.25.1 Description

The `TUnSortedStrCollection` object manages an unsorted list of strings. To this end, it overrides the TSortedCollection.Insert (1232) method to add strings at the end of the collection, rather than in the alphabetically correct position.

Take care, the Search (1231) and IndexOf (1197) methods will not work on an unsorted string collection.

67.25.2 Method overview

Page	Method	Description
1249	Insert	Insert a new string in the collection.

67.25.3 TUnSortedStrCollection.Insert

Synopsis: Insert a new string in the collection.

Declaration: `procedure Insert(Item: Pointer); Virtual`

Visibility: default

Description: `Insert` inserts a string at the end of the collection, instead of on it's alphabetical place, resulting in an unsorted collection of strings.

Errors: None.

See also: TCollection.Insert (1203)

Listing: ./examples/objectex/ex39.pp

```
Program ex39;

{ Program to demonstrate the TUnsortedStrCollection.Insert method }

Uses Objects,Strings;

Var C : PUnsortedStrCollection;
    S : String;
    I : longint;
    P : Pchar;

begin
  Randomize;
  C:=New(PUnsortedStrCollection,Init(120,10));
  Writeln ('Inserting 100 records at random places.');
  For I:=1 to 100 do
    begin
    Str(Random(100),S);
    S:='String with value '+S;
    C^.Insert(NewStr(S));
    end;
  For I:=0 to 99 do
    Writeln (I:2,': ',PString(C^.At(i))^ );
  Dispose(C,Done);
end.
```

Chapter 68

Reference for unit 'objpas'

68.1 Used units

Table 68.1: Used units by unit 'objpas'

Name	Page
System	1362

68.2 Overview

The objpas unit is meant for compatibility with Object Pascal as implemented by Delphi. The unit is loaded automatically by the Free Pascal compiler whenever the `Delphi` or `objfpc` mode is entered, either through the command line switches `-Sd` or `-Sh` or with the `{$MODE DELPHI}` or `{$MODE OBJFPC}` directives.

It redefines some basic pascal types, introduces some functions for compatibility with Delphi's system unit, and introduces some methods for the management of the resource string tables.

68.3 Constants, types and variables

68.3.1 Constants

`MaxInt = MaxLongint`

Maximum value for Integer (1252) type.

68.3.2 Types

`FixedInt = Int32`

`FixedInt` is provided for Delphi compatibility.

`FixedUInt = UInt32`

FixedUInt is provided for Delphi compatibility.

```
Integer = LongInt
```

In OBJPAS mode and in DELPHI mode, an Integer has a size of 32 bit. In TP or regular FPC mode, an integer is 16 bit.

```
IntegerArray = Array[0..$efffffff] of Integer
```

Generic array of integer (1252).

```
PFixedInt = ^FixedInt
```

```
PFixedUInt = ^FixedUInt
```

```
PInteger = ^Integer
```

Pointer to Integer (1252) type.

```
PIntegerArray = ^IntegerArray
```

Pointer to TIntegerArray (1252) type.

```
PointerArray = Array[0..512*1024*1024-2] of Pointer
```

Generic Array of pointers.

```
PPointerArray = ^PointerArray
```

Pointer to PointerArray (1252).

```
PString = PAnsiString
```

Pointer to ansistring type.

```
TEndian = (Little,Big)
```

Table 68.2: Enumeration values for type TEndian

Value	Explanation
Big	Big endian byte order.
Little	Little endian byte order.

TEndian describes the endianness of a computer architecture.

```
TIntegerArray = IntegerArray
```

Alias for IntegerArray (1252).

```
TPointerArray = PointerArray
```

Alias for PointerArray (1252).

Chapter 69

Reference for unit 'ports'

69.1 Used units

Table 69.1: Used units by unit 'ports'

Name	Page
System	1362

69.2 Overview

The ports unit implements the `port` constructs found in Turbo Pascal. It uses classes and default array properties to do this.

The unit exists on Linux, OS/2 and Dos. It is implemented only for compatibility with Turbo Pascal. Its usage is discouraged, because using ports is not portable programming, and the operating system may not even allow it (for instance Windows).

Under Linux, your program must be run as root, or the `IOPerm` call must be set in order to set appropriate permissions on the port access.

69.3 Constants, types and variables

69.3.1 Variables

```
port : tport
```

Default instance of type TPort (1254). Do not free. This variable is initialized in the unit initialization code, and freed at finalization.

Since there is a default property for a variable of this type, a sentence as

```
port[221]:=12;
```

Will result in the integer 12 being written to port 221, if port is defined as a variable of type `tport`

```
portb : tport
```

Default instance of type TPort (1254). Do not free. This variable is initialized in the unit initialization code, and freed at finalization.

Since there is a default property for a variable of this type, a sentence as

```
portb[221]:=12;
```

Will result in the byte 12 being written to port 221, if port is defined as a variable of type `tport`

```
portl : tportl
```

Default instance of type TPortL (1255). Do not free. This variable is initialized in the unit initialization code, and freed at finalization.

Since there is a default property for a variable of this type, a sentence as

```
portl[221]:=12;
```

Will result in the longint 12 being written to port 221, if port is defined as a variable of type `tport`

```
portw : tportw
```

Default instance of type TPortW (1255). Do not free. This variable is initialized in the unit initialization code, and freed at finalization.

Since there is a default property for a variable of this type, a sentence as

```
portw[221]:=12;
```

Will result in the word 12 being written to port 221, if port is defined as a variable of type `tport`

69.4 tport

69.4.1 Description

The `TPort` type is implemented specially for access to the ports in a TP compatible manner. There is no need to create an instance of this type: the standard TP variables are instantiated at unit initialization.

See also: port (1253), TPortW (1255), TPortL (1255)

69.4.2 Property overview

Page	Properties	Access	Description
1254	pp	rw	Access integer-sized port by port number.

69.4.3 tport.pp

Synopsis: Access integer-sized port by port number.

Declaration: `Property pp[w: Word]: Byte; default`

Visibility: public

Access: Read,Write

Description: Access integer-sized port by port number.

69.5 tportl

69.5.1 Description

The `TPortL` type is implemented specially for access to the ports in a TP compatible manner. There is no need to create an instance of this type: the standard TP variables are instantiated at unit initialization.

See also: portw (1254), TPort (1254), TPortL (1255)

69.5.2 Property overview

Page	Properties	Access	Description
1255	pp	rw	Access Longint-sized port by port number.

69.5.3 tportl.pp

Synopsis: Access Longint-sized port by port number.

Declaration: `Property pp[w: Word]: LongInt; default`

Visibility: public

Access: Read,Write

Description: Access Longint-sized port by port number.

69.6 tportw

69.6.1 Description

The `TPortW` type is implemented specially for access to the ports in a TP compatible manner. There is no need to create an instance of this type: the standard TP variables are instantiated at unit initialization.

See also: portw (1254), TPort (1254), TPortL (1255)

69.6.2 Property overview

Page	Properties	Access	Description
1255	pp	rw	Access word-sized port by port number.

69.6.3 tportw.pp

Synopsis: Access word-sized port by port number.

Declaration: `Property pp[w: Word]: Word; default`

Visibility: public

Access: Read,Write

Description: Access word-sized port by port number.

Chapter 70

Reference for unit 'printer'

70.1 Used units

Table 70.1: Used units by unit 'printer'

Name	Page
System	1362

70.2 Overview

This chapter describes the printer unit for Free Pascal. It was written for DOS by Florian Klaempfl, and it was written for Linux by Michael Van Canneyt, and has been ported to Windows and OS/2 as well. Its basic functionality is the same for all supported systems, although there are minor differences on Linux and UNIX.

70.3 Constants, types and variables

70.3.1 Variables

```
Lst : text
```

`Lst` is the standard printing device.
On Linux, `Lst` is set up using `AssignLst('/tmp/PID.lst')`.

70.4 Procedures and functions

70.4.1 AssignLst

Synopsis: Assign text file to printing device.

Declaration: `procedure AssignLst(var F: text; ToFile: string)`

Visibility: default

Description: `AssignLst` assigns to `F` a printing device - *UNIX only*. `ToFile` is a string with the following form:

- `'|filename options'`: This sets up a pipe with the program filename, with the given options, such as in the popen() call.
- `'filename'`: Prints to file filename. Filename can contain the string 'PID' (No Quotes), which will be replaced by the PID of your program. When closing lst, the file will be sent to lpr and deleted. (lpr should be in PATH)
- `{'filename|'}`: Same as previous, only the file is *not* sent to lpr, nor is it deleted. (useful for opening /dev/printer or for later printing)

See also: lst (1256)

Listing: ./examples/printex/printex.pp

```
program testprn;

uses printer;

var i : integer;
    f : text;

begin
  writeln ('Test of printer unit');
  writeln ('Writing to lst...');
  for i:=1 to 80 do writeln (lst,'This is line ',i,'.'#13);
  close (lst);
  writeln ('Done.');
  {$ifdef Unix}
  writeln ('Writing to pipe...');
  assignlst (f,'|/usr/bin/lpr -m');
  rewrite (f);
  for i:=1 to 80 do writeln (f,'This is line ',i,'.'#13);
  close (f);
  writeln ('Done.')
  {$endif}
end.
```

70.4.2 InitPrinter

Synopsis: Initialize the printer.

Declaration: `procedure InitPrinter(const PrinterName: string)`

Visibility: default

Description: Initialize the printer.

70.4.3 IsLstAvailable

Synopsis: Determine whether printer is available.

Declaration: `function IsLstAvailable : Boolean`

Visibility: default

Description: Determine whether printer is available.

Chapter 71

Reference for unit 'sharemem'

71.1 Used units

Table 71.1: Used units by unit 'sharemem'

Name	Page
System	1362

71.2 Overview

`sharemem` implements a shared memory manager. Including this unit will replace the standard memory manager with a memory manager which uses shared memory. This means the memory allocated by this unit can be managed by a program and a DLL if they both use the shared memory manager: it allows, amongst other things, to pass ansistrings or Unicode strings from a program to a DLL and vice versa.

This unit does not implement any routines: all actions to replace the memory manager are performed in the initialization section of the unit. The unit should be placed as the first unit in a program or DLL's uses section, memory corruption may occur if the unit is not placed first.

This unit requires the fpcmemdll.dll library to be distributed with both program and dll that use this unit. This DLL is distributed with the windows Free Pascal distribution.

Chapter 72

Reference for unit 'Sockets'

72.1 Used units

Table 72.1: Used units by unit 'Sockets'

Name	Page
BaseUnix	146
System	1362
unixtype	2175

72.2 Overview

This document describes the SOCKETS unit for Free Pascal. it was written for Linux by Michael Van Canneyt, and ported to Windows by Florian Klaempfl.

72.3 Constants, types and variables

72.3.1 Constants

`AF_ALG = 38`

Address family: Algorithm sockets.

`AF_APPLETALK = 5`

Address family Appletalk DDP.

`AF_ASH = 18`

Address family: Ash.

`AF_ATMPVC = 8`

Address family: ATM PVCs.

```
AF_ATMSVC = 20
```

Address family: ATM SVCs.

```
AF_AX25 = 3
```

Address family Amateur Radio AX.25.

```
AF_BLUETOOTH = 31
```

Address family: Bluetooth sockets.

```
AF_BRIDGE = 7
```

Address family Multiprotocol bridge.

```
AF_CAIF = 37
```

Address family: CAIF (Application CPU interface) sockets.

```
AF_CAN = 29
```

Address family: CAN (Controller Area Network) sockets.

```
AF_DECnet = 12
```

Address family: Reserved for DECnet project.

```
AF_ECONET = 19
```

Address family: Acorn Econet.

```
AF_IB = 27
```

Address family: IB (Infiniband) sockets.

```
AF_IEEE802154 = 36
```

Address family: IEEE802154 sockets.

```
AF_INET = 2
```

Address family Internet IP Protocol.

```
AF_INET6 = 10
```

Address family IP version 6.

```
AF_IPX = 4
```

Address family Novell IPX.

```
AF_IRDA = 23
```

Address family: IRDA sockets.

```
AF_ISDN = 34
```

Address family: ?

```
AF_IUCV = 32
```

Address family: IUCV (Inter User Communication Vehicle) sockets.

```
AF_KCM = 41
```

Address family: KCM (Kernel Connection Multiplexor) sockets.

```
AF_KEY = 15
```

Address family: PF_KEY key management API.

```
AF_LLC = 26
```

Address family: Linux LLC.

```
AF_LOCAL = 1
```

Address family: Unix socket.

```
AF_MAX = 45
```

Address family Maximum value.

```
AF_MPLS = 28
```

Address family: MPLS (Multiprotocol Label Switching) sockets.

```
AF_NETBEUI = 13
```

Address family: Reserved for 802.2LLC project.

```
AF_NETLINK = 16
```

Address family: ?

```
AF_NETROM = 6
```

Address family Amateur radio NetROM.

```
AF_NFC = 39
```

Address family: NFC (Near Field Communication) sockets.

```
AF_PACKET = 17
```

Address family: Packet family.

```
AF_PHONET = 35
```

Address family: Phonet sockets.

```
AF_PPPOX = 24
```

Address family: PPPoX sockets.

```
AF_QIPCRTR = 42
```

Address family: QIPCRTR (Qualcomm IPC Router) sockets.

```
AF_RDS = 21
```

Address Family: RDS (Reliable Datagram Sockets) sockets.

```
AF_ROSE = 11
```

Address family: Amateur Radio X.25 PLP.

```
AF_ROUTE = AF_NETLINK
```

Address family: Alias to emulate 4.4BSD.

```
AF_RXRPC = 33
```

Address family: RxRPC sockets.

```
AF_SECURITY = 14
```

Address family: Security callback pseudo AF.

```
AF_SMC = 43
```

Address family: SMC (Shared Memory Communication) sockets.

```
AF_SNA = 22
```

Address family: Linux SNA project.

```
AF_TIPC = 30
```

Address family: TIPC sockets.

```
AF_UNIX = 1
```

Address family Unix domain sockets.

```
AF_UNSPEC = 0
```

Address family Not specified.

```
AF_VSOCK = 40
```

Address family: VSOCK (hypervisor) sockets.

```
AF_WANPIPE = 25
```

Address family: Wanpipe API Sockets.

```
AF_X25 = 9
```

Address family Reserved for X.25 project.

```
AF_XDP = 44
```

Address family: XDP (Express Data Path) sockets.

```
EsockADDRINUSE = ESysEADDRINUSE
```

`EsockADDRINUSE` is the error reported by fpBind (1295) when the socket is already in use.

```
EsockEACCESS = ESysEAcces
```

Access forbidden error.

```
EsockEBADF = EsysEBADF
```

Alias: bad file descriptor.

```
EsockEFAULT = EsysEFAULT
```

Alias: an error occurred.

```
EsockEINTR = EsysEINTR
```

Alias : operation interrupted.

```
EsockEINVAL = EsysEINVAL
```

Alias: Invalid value specified.

```
EsockEMFILE = ESysEmfile
```

Error code ?

```
EsockEMSGSIZE = ESysEMsgSize
```

Wrong message size error.

```
EsockENOBUFS = ESysENoBufs
```

No buffer space available error.

```
EsockENOTCONN = ESysENotConn
```

Not connected error.

```
EsockENOTSOCK = ESysENotSock
```

File descriptor is not a socket error.

```
EsockEPROTONOSUPPORT = ESysEProtoNoSupport
```

Protocol not supported error.

```
EsockEWOULDBLOCK = ESysEWouldBlock
```

Operation would block error.

```
INADDR_ANY = CARDINAL(0)
```

A bitmask matching any IP address on the local machine.

```
INADDR_NONE = CARDINAL($FFFFFFFF)
```

A bitmask matching no valid IP address.

```
IPPROTO_AH = 51
```

authentication header.

```
IPPROTO_COMP = 108
```

Compression Header Protocol.

```
IPPROTO_DSTOPTS = 60
```

IPv6 destination options.

```
IPPROTO_EGP = 8
```

Exterior Gateway Protocol.

```
IPPROTO_ENCAP = 98
```

Encapsulation Header.

```
IPPROTO_ESP = 50
```

encapsulating security payload.

```
IPPROTO_FRAGMENT = 44
```

IPv6 fragmentation header.

```
IPPROTO_GRE = 47
```

General Routing Encapsulation.

```
IPPROTO_HOPOPTS = 0
```

IPv6 Hop-by-Hop options.

```
IPPROTO_ICMP = 1
```

Internet Control Message Protocol.

```
IPPROTO_ICMPV6 = 58
```

ICMPv6.

```
IPPROTO_IDP = 22
```

XNS IDP protocol.

```
IPPROTO_IGMP = 2
```

Internet Group Management Protocol.

```
IPPROTO_IP = 0
```

Dummy protocol for TCP.

```
IPPROTO_IPIP = 4
```

IPIP tunnels (older KA9Q tunnels use 94).

```
IPPROTO_IPV6 = 41
```

IPv6 header.

```
IPPROTO_MAX = 255
```

Maximum value for IPPROTO options.

```
IPPROTO_MTP = 92
```

Multicast Transport Protocol.

```
IPPROTO_NONE = 59
```

IPv6 no next header.

```
IPPROTO_PIM = 103
```

Protocol Independent Multicast.

```
IPPROTO_PUP = 12
```

PUP protocol.

```
IPPROTO_RAW = 255
```

Raw IP packets.

```
IPPROTO_ROUTING = 43
```

IPv6 routing header.

```
IPPROTO_RSVP = 46
```

Reservation Protocol.

```
IPPROTO_SCTP = 132
```

Stream Control Transmission Protocol.

```
IPPROTO_TCP = 6
```

Transmission Control Protocol.

```
IPPROTO_TP = 29
```

SO Transport Protocol Class 4.

```
IPPROTO_UDP = 17
```

User Datagram Protocol.

```
IPV6_ADDRFORM = 1
```

Change the IPV6 address into a different address family. Deprecated.

```
IPV6_ADD_MEMBERSHIP = IPV6_JOIN_GROUP
```

Undocumented Getsockopt option ?

```
IPV6_AUTHHDR = 10
```

GetSockOpt/SetSockopt: Deliver authentication header messages.

```
IPV6_CHECKSUM = 7
```

Undocumented Getsockopt option ?

```
IPV6_DROP_MEMBERSHIP = IPV6_LEAVE_GROUP
```

Undocumented Getsockopt option ?

```
IPV6_DSTOPTS = 59
```

Deliver destination option control messages.

```
IPV6_HOPLIMIT = 52
```

Deliver an integer containing the HOP count.

```
IPV6_HOPOPTS = 54
```

Deliver hop option control messages.

```
IPV6_IPSEC_POLICY = 34
```

Undocumented Getsockopt option ?

```
IPV6_JOIN_ANYCAST = 27
```

Undocumented Getsockopt option ?

```
IPV6_JOIN_GROUP = 20
```

GetSockOpt/SetSockopt: Control membership (join group) in multicast groups.

```
IPV6_LEAVE_ANYCAST = 28
```

Undocumented Getsockopt option ?

```
IPV6_LEAVE_GROUP = 21
```

GetSockOpt/SetSockopt: Control membership (leave group)in multicast groups.

```
IPV6_MTU = 24
```

GetSockOpt/SetSockopt: Get/Set the MTU for the socket.

```
IPV6_MTU_DISCOVER = 23
```

GetSockOpt/SetSockopt: Get/Set Control path MTU Discovery on the socket.

```
IPV6_MULTICAST_HOPS = 18
```

GetSockOpt/SetSockopt: Get/Set the multicast hop limit.

```
IPV6_MULTICAST_IF = 17
```

GetSockOpt/SetSockopt: Get/Set device for multicast packages on the socket.

```
IPV6_MULTICAST_LOOP = 19
```

GetSockOpt/SetSockopt: Control whether socket sees multicast packages that it has sent itself.

```
IPV6_NEXTHOP = 9
```

sendmsg: set next hop for IPV6 datagram.

```
IPV6_PKTINFO = 50
```

Change delivery options for incoming IPV6 datagrams.

```
IPV6_PKTOPTIONS = 6
```

Undocumented Getsockopt option ?

```
IPV6_PMTUDISC_DO = 2
```

Always DF.

```
IPV6_PMTUDISC_DONT = 0
```

Never send DF frames.

```
IPV6_PMTUDISC_WANT = 1
```

Use per route hints.

```
IPV6_RECVERR = 25
```

GetSockOpt/SetSockopt: Control receiving of asynchronous error options.

```
IPV6_ROUTER_ALERT = 22
```

GetSockOpt/SetSockopt: Get/Set Pass all forwarded packets containing router alert option.

```
IPV6_RTHDR = 57
```

Deliver routing header control messages.

```
IPV6_RTHDR_LOOSE = 0
```

Hop doesn't need to be neighbour.

```
IPV6_RTHDR_STRICT = 1
```

Hop must be a neighbour.

```
IPV6_RTHDR_TYPE_0 = 0
```

IPv6 Routing header type 0.

```
IPV6_RXDSTOPTS = IPV6_DSTOPTS
```

Undocumented Getsockopt option ?

```
IPV6_RXHOPOPTS = IPV6_HOPOPTS
```

Undocumented Getsockopt option ?

```
IPV6_RXSRCRT = IPV6_RTHDR
```

Undocumented Getsockopt option ?

```
IPV6_UNICAST_HOPS = 16
```

GetSockOpt/SetSockopt: Get/Set unicast hop limit.

```
IPV6_V6ONLY = 26
```

GetSockOpt/SetSockopt: Handle IPV6 connections only.

```
IPV6_XFRM_POLICY = 35
```

Undocumented Getsockopt option ?

```
IPX_TYPE = 1
```

IPX type.

```
IP_ADD_MEMBERSHIP = 35
```

add an IP group membership.

```
IP_ADD_SOURCE_MEMBERSHIP = 39
```

join source group.

```
IP_BLOCK_SOURCE = 38
```

block data from source.

```
IP_DEFAULT_MULTICAST_LOOP = 1
```

Undocumented ?

```
IP_DEFAULT_MULTICAST_TTL = 1
```

Undocumented ?

```
IP_DROP_MEMBERSHIP = 36
```

drop an IP group membership.

```
IP_DROP_SOURCE_MEMBERSHIP = 40
```

leave source group.

```
IP_HDRINCL = 3
```

Header is included with data.

```
IP_MAX_MEMBERSHIPS = 20
```

Maximum group memberships for multicast messages.

```
IP_MSFILTER = 41
```

Undocumented ?

```
IP_MTU_DISCOVER = 10
```

Undocumented ?

```
IP_MULTICAST_IF = 32
```

set/get IP multicast i/f.

```
IP_MULTICAST_LOOP = 34
```

set/get IP multicast loopback.

```
IP_MULTICAST_TTL = 33
```

set/get IP multicast ttl.

```
IP_OPTIONS = 4
```

IP per-packet options.

```
IP_PKTINFO = 8
```

Undocumented ?

```
IP_PKTOPTIONS = 9
```

Undocumented ?

```
IP_PMTUDISC = 10
```

Undocumented ?

```
IP_PMTUDISC_DO = 2
```

Always DF.

```
IP_PMTUDISC_DONT = 0
```

Never send DF frames.

```
IP_PMTUDISC_WANT = 1
```

Use per route hints.

```
IP_RECVERR = 11
```

Undocumented ?

```
IP_RECVOPTS = 6
```

Receive all IP options w/datagram.

```
IP_RECVRETOPTS = IP_RETOPTS
```

Receive IP options for response.

```
IP_RECVTOS = 13
```

Undocumented ?

```
IP_RECVTTL = 12
```

Undocumented ?

```
IP_RETOPTS = 7
```

Set/get IP per-packet options.

```
IP_ROUTER_ALERT = 5
```

Undocumented ?

```
IP_TOS = 1
```

IP type of service and precedence.

```
IP_TTL = 2
```

IP time to live.

```
IP_UNBLOCK_SOURCE = 37
```

unblock data from source.

```
MCAST_BLOCK_SOURCE = 43
```

block from given group.

```
MCAST_EXCLUDE = 0
```

Undocumented ?

```
MCAST_INCLUDE = 1
```

Undocumented ?

```
MCAST_JOIN_GROUP = 42
```

join any-source group.

```
MCAST_JOIN_SOURCE_GROUP = 46
```

join source-spec gruoup.

```
MCAST_LEAVE_GROUP = 45
```

leave any-source group.

```
MCAST_LEAVE_SOURCE_GROUP = 47
```

leave source-spec group.

```
MCAST_MSFILTER = 48
```

Undocumented ?

```
MCAST_UNBLOCK_SOURCE = 44
```

unblock from given group.

```
MSG_BATCH = $00040000
```

sendmmsg: more messages coming.

```
MSG_CMSG_CLOEXEC = $40000000
```

Set close_on_exec for file descriptor received through SCM_RIGHTS.

```
MSG_CMSG_COMPAT = $0
```

This message needs 32 bit fixups.

```
MSG_CONFIRM = $00000800
```

Send flags: Conform connection.

```
MSG_CTRUNC = $00000008
```

Receive flags: Control Data was discarded (buffer too small).

```
MSG_DONTROUTE = $00000004
```

Send flags: don't use gateway.

```
MSG_DONTWAIT = $00000040
```

Receive flags: Non-blocking operation request.

```
MSG_EOF = MSG_FIN
```

Alias for `MSG_FIN`.

```
MSG_EOR = $00000080
```

Receive flags: End of record.

```
MSG_ERRQUERE = $00002000
```

Receive flags: ?

```
MSG_FASTOPEN = $20000000
```

Send data in TCP SYN.

```
MSG_FIN = $00000200
```

Receive flags: ?

```
MSG_MORE = $00008000
```

Receive flags: ?

```
MSG_NOSIGNAL = $00004000
```

Receive flags: Suppress SIG_PIPE signal.

```
MSG_NO_SHARED_FRAGS = $00080000
```

sendpage internal : page frags are not shared.

```
MSG_OOB = $00000001
```

Receive flags: receive out-of-band data.

```
MSG_PEEK = $00000002
```

Receive flags: peek at data, don't remove from buffer.

```
MSG_PROBE = MSG_PROXY
```

Do not send. Only probe path f.e. for MTU.

```
MSG_PROXY = $00000010
```

Receive flags: ?

```
MSG_RST = $00001000
```

Receive flags: ?

```
MSG_SENDPAGE_DECRYPTED = $00100000
```

sendpage() internal : page may carry plain text and require encryption.

```
MSG_SENDPAGE_NOPOLICY = $00010000
```

sendpage internal : do not apply policy.

```
MSG_SENDPAGE_NOTLAST = $00020000
```

sendpage internal : not the last page.

```
MSG_SYN = $00000400
```

Receive flags: ?

```
MSG_TRUNC = $00000020
```

Receive flags: packet Data was discarded (buffer too small).

```
MSG_TRYHARD = MSG_DONTROUTE
```

Receive flags: ?

```
MSG_WAITALL = $00000100
```

Receive flags: Wait till operation completed.

```
MSG_WAITFORONE = $00010000
```

recvmmsg(): block until 1+ packets avail.

```
MSG_ZEROCOPY = $04000000
```

Use user data in kernel path.

```
NoAddress : in_addr = (s_addr: 0)
```

Constant indicating invalid (no) network address.

```
NoAddress6 : in6_addr = (u6_addr16: (0, 0, 0, 0, 0, 0, 0, 0))
```

Constant indicating invalid (no) IPV6 network address.

```
NoNet : in_addr = (s_addr: 0)
```

Constant indicating invalid (no) network address.

```
NoNet6 : in6_addr = (u6_addr16: (0, 0, 0, 0, 0, 0, 0, 0))
```

Constant indicating invalid (no) IPV6 network address.

```
PF_ALG = AF_ALG
```

Protocol family: Algorithm sockets.

```
PF_APPLETALK = AF_APPLETALK
```

Protocol family: Appletalk DDP.

```
PF_ASH = AF_ASH
```

Protocol family: Ash.

```
PF_ATMPVC = AF_ATMPVC
```

Protocol family: ATM PVCs.

```
PF_ATMSVC = AF_ATMSVC
```

Protocol family: ATM SVCs.

```
PF_AX25 = AF_AX25
```

Protocol family: Amateur Radio AX.25.

```
PF_BLUETOOTH = AF_BLUETOOTH
```

Protocol family: Bluetooth sockets.

```
PF_BRIDGE = AF_BRIDGE
```

Protocol family: Multiprotocol bridge.

```
PF_CAIF = AF_CAIF
```

Protocol family: CAIF (Application CPU interface) sockets.

```
PF_CAN = AF_CAN
```

Protocol family: CAN (Controller Area Network) sockets.

```
PF_DECnet = AF_DECnet
```

Protocol Family: DECNET project.

```
PF_ECONET = AF_ECONET
```

Protocol family: Acorn Econet.

```
PF_IB = AF_IB
```

Protocol family: IB (InfiniBand) sockets.

```
PF_IEEE802154 = AF_IEEE802154
```

Protocol family: IEEE802154 sockets.

```
PF_INET = AF_INET
```

Protocol family: Internet IP Protocol.

```
PF_INET6 = AF_INET6
```

Protocol family: IP version 6.

```
PF_IPX = AF_IPX
```

Protocol family: Novell IPX.

```
PF_IRDA = AF_IRDA
```

Protocol family: IRDA sockets.

```
PF_ISDN = AF_ISDN
```

Protocol Family: ?

```
PF_IUCV = AF_IUCV
```

Protocol family: IUCV (Inter User Communication Vehicle) sockets.

```
PF_KCM = AF_KCM
```

Protocol family: KCM (Kernel Connection Multiplexor) sockets.

```
PF_KEY = AF_KEY
```

Protocol family: Key management API.

```
PF_LLC = AF_LLC
```

Protocol family: Linux LLC.

```
PF_LOCAL = AF_LOCAL
```

Protocol family: Unix socket.

```
PF_MAX = AF_MAX
```

Protocol family: Maximum value.

```
PF_MPLS = AF_MPLS
```

Protocol family: MPLS (Multiprotocol Label Switching.

```
PF_NETBEUI = AF_NETBEUI
```

Protocol family: Reserved for 802.2LLC project.

```
PF_NETLINK = AF_NETLINK
```

Protocol family: ?

```
PF_NETROM = AF_NETROM
```

Protocol family:Amateur radio NetROM.

```
PF_NFC = AF_NFC
```

Protocol family: NFC (Near Field Communication) sockets.

```
PF_PACKET = AF_PACKET
```

Protocol family: Packet family.

```
PF_PHONET = AF_PHONET
```

Protocol family: RxRPC sockets.

```
PF_PPPOX = AF_PPPOX
```

Protocol family: PPPoX sockets.

```
PF_QIPCRTR = AF_QIPCRTR
```

Protocol family: IPCRTR (Qualcomm IPC Router) sockets.

```
PF_RDS = AF_RDS
```

Protocol family: RDS (Reliable Datagram Sockets) sockets.

```
PF_ROSE = AF_ROSE
```

Protocol family: Amateur Radio X.25 PLP.

```
PF_ROUTE = AF_ROUTE
```

Protocol Family: ?

```
PF_RXRPC = AF_RXRPC
```

Protocol family: RxRPC sockets.

```
PF_SECURITY = AF_SECURITY
```

Protocol family: Security callback pseudo PF.

```
PF_SMC = AF_SMC
```

Protocol family: SMC (Shared Memory Communication) sockets.

```
PF_SNA = AF_SNA
```

Protocol Family: Linux SNA project.

```
PF_TIPC = AF_TIPC
```

Protocol family: TIPC sockets.

```
PF_UNIX = AF_UNIX
```

Protocol family: Unix domain sockets.

```
PF_UNSPEC = AF_UNSPEC
```

Protocol family: Unspecified.

```
PF_VSOCK = AF_VSOCK
```

Protocol family: VSOCK (hypervisor) sockets.

```
PF_WANPIPE = AF_WANPIPE
```

Protocol family: Wanpipe API Sockets.

```
PF_X25 = AF_X25
```

Protocol family: Reserved for X.25 project.

```
PF_XDP = AF_XDP
```

Protocol family: XDP (Express Data Path) sockets.

```
SCM_CREDENTIALS = $02
```

socket control messages : credentials.

```
SCM_RIGHTS = $01
```

socket control messages : access rights.

```
SCM_SECURITY = $03
```

socket control messages : security label.

```
SCM_SRCRT = IPV6_RXSRCRT
```

Undocumented Getsockopt option ?

```
SCM_TIMESTAMP = SO_TIMESTAMP
```

Socket option: ?

```
SHUT_RD = 0
```

Shutdown read part of full duplex socket.

```
SHUT_RDWR = 2
```

Shutdown read and write part of full duplex socket.

```
SHUT_WR = 1
```

Shutdown write part of full duplex socket.

```
SOCK_DGRAM = 2
```

Type of socket: datagram (conn.less) socket (UDP).

```
SOCK_MAXADDRLEN = 255
```

Maximum socket address length for Bind (1291) call.

```
SOCK_RAW = 3
```

Type of socket: raw socket.

```
SOCK_RDM = 4
```

Type of socket: reliably-delivered message.

```
SOCK_SEQPACKET = 5
```

Type of socket: sequential packet socket.

```
SOCK_STREAM = 1
```

Type of socket: stream (connection) type socket (TCP).

```
SOL_AAL = 265
```

Socket level: AAL.

```
SOL_ALG = 279
```

Socket level: Algorithm socket.

```
SOL_ATALK = 258
```

Socket level: Apple talk.

```
SOL_ATM = 264
```

Socket level: ATM.

```
SOL_AX25 = 257
```

Socket level: AX25.

```
SOL_BLUETOOTH = 274
```

Socket level: BlueTooth.

```
SOL_CAIF = 278
```

Socket level: CAIF (Application CPU interface).

```
SOL_DCCP = 269
```

Socket level: DCCP.

```
SOL_DECNET = 261
```

Socket level: DECnet.

```
SOL_ICMPV6 = 58
```

Socket level values for IPv6: ICMPV6.

```
SOL_IP = 0
```

Undocumented ?

```
SOL_IPV6 = 41
```

Socket level values for IPv6: IPV6.

```
SOL_IPX = 256
```

Socket level: IPX.

```
SOL_IRDA = 266
```

Socket level: IRDA.

```
SOL_IUCV = 277
```

Socket level: IUCV (Inter User Communication Vehicle).

```
SOL_KCM = 281
```

Socket level: KCM (Kernel Connection Multiplexor) sockets.

```
SOL_LLC = 268
```

Socket level: LLC.

```
SOL_NETBEUI = 267
```

Socket level: NETBEUI.

```
SOL_NETLINK = 270
```

Socket level: NETLink.

```
SOL_NETROM = 259
```

Socket level: Net ROM.

```
SOL_NFC = 280
```

Socket level: NFC (Near Field Communication) socket.

```
SOL_PACKET = 263
```

Socket level: Packet.

```
SOL_PNPIPE = 275
```

Socket level: PNPIPE.

```
SOL_PPPOL2TP = 273
```

Socket level: PPPOL2TP.

```
SOL_RAW = 255
```

Socket level: raw socket.

```
SOL_RDS = 276
```

Socket level: RDS (Reliable Datagram Socket).

```
SOL_ROSE = 260
```

Socket level: ROSE.

```
SOL_RXRPC = 272
```

Socket level: RxRPC.

```
SOL_SCTP = 132
```

Socket level: SCTP.

```
SOL_SOCKET = 1
```

Socket option level: Socket level.

```
SOL_TCP = 6
```

Socket level: TCP.

```
SOL_TIPC = 271
```

Socket level: TIPC.

```
SOL_TLS = 282
```

Socket level: TLS.

```
SOL_UDP = 17
```

Socket level: UDP.

```
SOL_UDPLITE = 136
```

Socket level: UDP Lite.

```
SOL_X25 = 262
```

Socket level: X25.

```
SOL_XDP = 283
```

Socket level: XDP (Express Data Path) sockets.

```
SOMAXCONN = 4096
```

Maximum queue length specifiable by listen.

```
SO_ACCEPTCONN = 30
```

Socket option: ?

```
SO_ATTACH_FILTER = 26
```

Socket option: ?

```
SO_BINDTODEVICE = 25
```

Socket option: ?

```
SO_BROADCAST = 6
```

Socket option: Broadcast.

```
SO_BSDCOMPAT = 14
```

Socket option: ?

```
SO_DEBUG = 1
```

Socket option level: debug.

```
SO_DETACH_FILTER = 27
```

Socket option: ?

```
SO_DONTROUTE = 5
```

Socket option: Don't route.

```
SO_ERROR = 4
```

Socket option: Error.

```
SO_KEEPALIVE = 9
```

Socket option: keep alive.

```
SO_LINGER = 13
```

Socket option: ?

```
SO_NO_CHECK = 11
```

Socket option: ?

```
SO_OOBINLINE = 10
```

Socket option: ?

```
SO_PASSCRED = 16
```

Socket option: ?

```
SO_PEERCRED = 17
```

Socket option: ?

```
SO_PEERNAME = 28
```

Socket option: ?

```
SO_PRIORITY = 12
```

Socket option: ?

```
SO_RCVBUF = 8
```

Socket option: receive buffer.

```
SO_RCVLOWAT = 18
```

Socket option: ?

```
SO_RCVTIMEO = 20
```

Socket option: ?

```
SO_REUSEADDR = 2
```

Socket option: Reuse address.

```
SO_REUSEPORT = 15
```

Socket Option : reuse port.

```
SO_SECURITY_AUTHENTICATION = 22
```

Socket option: ?

```
SO_SECURITY_ENCRYPTION_NETWORK = 24
```

Socket option: ?

```
SO_SECURITY_ENCRYPTION_TRANSPORT = 23
```

Socket option: ?

```
SO_SNDBUF = 7
```

Socket option: Send buffer.

```
SO_SNDLOWAT = 19
```

Socket option: ?

```
SO_SNDTIMEO = 21
```

Socket option: ?

```
SO_TIMESTAMP = 29
```

Socket option: ?

```
SO_TYPE = 3
```

Socket option: Type.

```
S_IN = 0
```

Input socket in socket pair.

```
S_OUT = 1
```

Output socket in socket pair.

```
TCP_CONGESTION = 13
```

Get/set the congestion-control algorithm for this socket.

```
TCP_CORK = 3
```

Get/Set CORK algorithm: Send only complete packets.

```
TCP_DEFER_ACCEPT = 9
```

Get/Set deferred accept on server socket.

```
TCP_INFO = 11
```

Get TCP connection information (Linux only).

```
TCP_KEEPCNT = 6
```

Get/Set retry count for unacknowledged KEEPALIVE transmissions.

```
TCP_KEEPIDLE = 4
```

Get/Set inactivity interval between KEEPALIVE transmissions.

```
TCP_KEEPINTVL = 5
```

Get/Set retry interval for unacknowledged KEEPALIVE transmissions.

```
TCP_LINGER2 = 8
```

Get/Set Linger2 flag.

```
TCP_MAXSEG = 2
```

Get/Set Maximum segment size.

```
TCP_MD5SIG = 14
```

Get/Set TCP MD5 signature option.

```
TCP_NODELAY = 1
```

Get/Set No delay flag: disable Nagle algorithm.

```
TCP_QUICKACK = 12
```

Get/Set quick ACK packet option.

```
TCP_SYNCNT = 7
```

Get/Set number of SYN packets to send before giving up on connection establishment.

```
TCP_WINDOW_CLAMP = 10
```

Get/Set maximum packet size.

```
UDP_CORK = 1
```

Get/Set UDP CORK algorithm on datagram sockets.

```
UDP_ENCAP = 100
```

Get/Set UDP encapsulation flag for IPSec datagram sockets.

```
UDP_ENCAP_ESPINUDP = 2
```

? Undocumented datagram option, IPSec related.

```
UDP_ENCAP_ESPINUDP_NON_IKE = 1
```

? Undocumented datagram option, IPSec related.

```
UDP_ENCAP_L2TPINUDP = 3
```

? Undocumented datagram option, IPSec related.

72.3.2 Types

```
in6_addr = packed record
case Byte of
0: (
  u6_addr8 : Array[0..15
  ] of Byte;
);
1: (
  u6_addr16 : Array[0..7] of Word;
);
```

```
2: (
  u6_addr32
  : Array[0..3] of Cardinal;
);
3: (
  s6_addr8 : Array[0..15] of ShortInt
  ;
);
4: (
  s6_addr : Array[0..15] of ShortInt;
);
5: (
  s6_addr16
  : Array[0..7] of SmallInt;
);
6: (
  s6_addr32 : Array[0..3] of LongInt
  ;
);
end
```

Record used to describe a general IPV6 address.

```
in_addr = packed record
case Boolean of
True: (
  s_addr : cuint32
  ;
);
False: (
  s_bytes : packed Array[1..4] of Byte;
);
end
```

General inet socket address.

```
PIn6Addr = pin6_addr
```

Pointer to in6_addr (1287) type.

```
pin6_addr = ^in6_addr
```

Pointer to Tin6_addr (1289).

```
PInAddr = pin_addr
```

Alias for pin_addr (1288).

```
PInetSockAddr = psockaddr_in
```

Pointer to sockaddr_in (1308).

```
PInetSockAddr6 = psockaddr_in6
```

Pointer to sockaddr_in6 (1308) type.

```
pin_addr = ^in_addr
```

Pointer to in_addr (1287) record.

```
plinger = ^linger
```

Pointer to linger (1308) type.

```
psockaddr = ^sockaddr
```

Pointer to TSockAddr (1289).

```
PSockAddr6 = ^TSockAddr6
```

`PSockAddr6` is a pointer to a record of type TSockAddr6 (1289).

```
psockaddr_in = ^sockaddr_in
```

Pointer to sockaddr_in (1308).

```
psockaddr_in6 = ^sockaddr_in6
```

Pointer to sockaddr_in6 (1308).

```
psockaddr_un = ^sockaddr_un
```

Pointer to sockaddr_un (1309) type.

```
Pucred = ^ucred
```

`Pucred` is a pointer to ucred (1309)

```
sa_family_t = cushort
```

Address family type.

```
sockaddr = packed record
case Integer of
0: (
  sa_family : sa_family_t
  ;
  sa_data : packed Array[0..13] of cuint8;
);
1: (
  sin_family
  : sa_family_t;
  sin_port : cushort;
  sin_addr : in_addr;
  sin_zero
  : packed Array[0..7] of cuint8;
);
end
```

sockaddr is used to store a general socket address for the FPBind (1295), FPRecv (1299) and FPSend (1300) calls.

```
TIn6Addr = in6_addr
```

Alias for in6_addr (1287) type.

```
Tin6_addr = in6_addr
```

Alias for sockaddr_in6 (1308).

```
TInAddr = in_addr
```

Alias for in_addr (1287) record type.

```
TInetSockAddr = sockaddr_in
```

Alias for sockaddr_in (1308).

```
TInetSockAddr6 = sockaddr_in6
```

Alias for sockaddr_in6 (1308).

```
TIn_addr = in_addr
```

Alias for in_addr (1287) record type.

```
TLinger = linger
```

Alias for linger (1308).

```
TSockAddr = sockaddr
```

```
TSockAddr6 = sockaddr_in6
```

TSockAddr6 is an alias for sockaddr6_in (1259)

```
TSockArray = Array[1..2] of LongInt
```

Type returned by the FPSocketPair (1302) call.

```
Tsocket = LongInt
```

Alias for easy kylix porting.

```
TSockLen = BaseUnix.TSocklen
```

The actual type of TSockLen depends on the platform.

```
TSockPairArray = Array[0..1] of LongInt
```

Array of sockets, used in FPSocketPair (1302) call.

72.4 Procedures and functions

72.4.1 Accept

Synopsis: Accept a connection from a socket (deprecated).

Declaration:
```
function Accept(Sock: LongInt; var addr: TInetSockAddr;
                var SockIn: File; var SockOut: File) : Boolean
function Accept(Sock: LongInt; var addr: TInetSockAddr;
                var SockIn: text; var SockOut: text) : Boolean
function Accept(Sock: LongInt; var addr: string; var SockIn: text;
                var SockOut: text) : Boolean
function Accept(Sock: LongInt; var addr: string; var SockIn: File;
                var SockOut: File) : Boolean
```

Visibility: default

Description: `Accept` accepts a connection from a socket `Sock`, which was listening for a connection. If a connection is accepted, a file descriptor is returned. On error `-1` is returned. The returned socket may NOT be used to accept more connections. The original socket remains open.

The `Accept` call fills the address of the connecting entity in `Addr`, and sets its length in `Addrlen`. `Addr` should be pointing to enough space, and `Addrlen` should be set to the amount of space available, prior to the call.

The alternate forms of the Accept (1290) command, with the `Text` or `File` parameters are equivalent to subsequently calling the regular Accept (1290) function and the Sock2Text (1306) or Sock2File (1305) functions. These functions return `True` if successful, `False` otherwise.

Errors: On error, `-1` is returned, and errors are reported in `SocketError`, and include the following:

ESockEBADF (1263) The socket descriptor is invalid.

ESockENOTSOCK (1264) The descriptor is not a socket.

SYS_EOPNOTSUPP The socket type doesn't support the `Listen` operation.

ESockEFAULT (1263)`Addr` points outside your address space.

ESockEWOULDBLOCK (1264) The requested operation would block the process.

See also: FPListen (1299), Connect (1292), FPConnect (1295), FPBind (1295)

Listing: ./examples/sockex/socksvr.pp

```
Program server;

{
  Program to test Sockets unit by Michael van Canneyt and Peter Vreman
  Server Version, First Run sock_svr to let it create a socket and then
  sock_cli to connect to that socket
}
{$mode fpc}
uses Sockets;

Var
  FromName : string;
  Buffer   : string[255];
  S        : Longint;
  Sin,Sout : Text;
  SAddr    : TInetSockAddr;
```

```
procedure perror (const S:string);
begin
  writeln (S,SocketError);
  halt(100);
end;

begin
  S:=fpSocket (AF_INET,SOCK_STREAM,0);
  if SocketError<>0 then
   Perror ('Server : Socket : ');
  SAddr.sin_family:=AF_INET;
  { port 50000 in network order }
  SAddr.sin_port:=htons(5000);
  SAddr.sin_addr.s_addr:=0;
  if fpBind(S,@SAddr,sizeof(saddr))=-1 then
   PError ('Server : Bind : ');
  if fpListen (S,1)=-1 then
   PError ('Server : Listen : ');
  Writeln('Waiting for Connect from Client, run now sock_cli in an other tty');
  if Accept(S,FromName,Sin,Sout) then
   PError ('Server : Accept : '+fromname);
  Reset(Sin);
  ReWrite(Sout);
  Writeln(Sout,'Message From Server');
  Flush(SOut);
  while not eof(sin) do
   begin
     Readln(Sin,Buffer);
     Writeln('Server : read : ',buffer);
   end;
end.
```

72.4.2 Bind

Synopsis: Alias for `fpBind`.

Declaration: `function Bind(Sock: LongInt; const addr: string) : Boolean`

Visibility: default

Description: `Bind` is an alias for fpBind (1295) which binds to either a socket struct (`Addr`) or a string indicating a UNIX socket file (UNIX socket).

This function is deprecated, use `fpBind` instead.

See also: FPBind (1295)

72.4.3 CloseSocket

Synopsis: Closes a socket handle.

Declaration: `function CloseSocket(Sock: LongInt) : LongInt`

Visibility: default

Description: `CloseSocket` closes a socket handle. It returns 0 if the socket was closed successfully, -1 if it failed.

Errors: On error, -1 is returned.

See also: FPSocket (1302)

72.4.4 Connect

Synopsis: Open a connection to a server socket (deprecated).

Declaration:
```
function Connect(Sock: LongInt; const addr: TInetSockAddr;
                 var SockIn: text; var SockOut: text) : Boolean
function Connect(Sock: LongInt; const addr: TInetSockAddr;
                 var SockIn: File; var SockOut: File) : Boolean
function Connect(Sock: LongInt; const addr: string; var SockIn: text;
                 var SockOut: text) : Boolean
function Connect(Sock: LongInt; const addr: string; var SockIn: File;
                 var SockOut: File) : Boolean
```

Visibility: default

Description: `Connect` opens a connection to a peer, whose address is described by `Addr`. `AddrLen` contains the length of the address. The type of `Addr` depends on the kind of connection you're trying to make, but is generally one of `TSockAddr` or `TUnixSockAddr`.

The forms of the Connect (1292) command with the `Text` or `File` arguments are equivalent to subsequently calling the regular `Connect` function and the Sock2Text (1306) or Sock2File (1305) functions. These functions return `True` if success full, `False` otherwise.

The `Connect` function returns a file descriptor if the call was success full, `-1` in case of error.

Errors: On error, `-1` is returned and errors are reported in `SocketError`.

See also: FPListen (1299), FPBind (1295), Accept (1290), FPAccept (1294)

Listing: ./examples/sockex/sockcli.pp

```
Program Client;

{
  Program to test Sockets unit by Michael van Canneyt and Peter Vreman
  Client Version, First Run sock_svr to let it create a socket and then
  sock_cli to connect to that socket
}

uses Sockets;

procedure PError(const S : string);
begin
  writeln(S,SocketError);
  halt(100);
end;


Var
  SAddr     : TInetSockAddr;
  Buffer    : string [255];
  S         : Longint;
  Sin,Sout  : Text;
  i         : integer;
```

```
begin
  S:=fpSocket (AF_INET,SOCK_STREAM,0);
  if s=-1 then
   Perror('Client : Socket : ');
  SAddr.sin_family:=AF_INET;
  { port 50000 in network order }
  SAddr.sin_port:=htons(5000);
  { localhost : 127.0.0.1 in network order }
  SAddr.sin_addr.s_addr:=HostToNet((127 shl 24) or 1);
  if not Connect (S,SAddr,Sin,Sout) then
   PError('Client : Connect : ');
  Reset(Sin);
  ReWrite(Sout);
  Buffer:='This is a textstring sent by the Client.';
  for i:=1 to 10 do
    WriteLn(Sout,Buffer);
  Flush(Sout);
  ReadLn(SIn,Buffer);
  WriteLn(Buffer);
  Close(sout);
end.
```

Listing: ./examples/sockex/pfinger.pp

```
program pfinger;

uses sockets,errors;

Var
  Addr : TInetSockAddr;
  S : Longint;
  Sin,Sout : Text;
  Line : string;

begin
  Addr.sin_family:=AF_INET;
  { port 79 in network order }
  Addr.sin_port:=79 shl 8;
  { localhost : 127.0.0.1 in network order }
  Addr.sin_addr.s_addr:=((1 shl 24) or 127);
  S:=fpSocket(AF_INET,SOCK_STREAM,0);
  If Not Connect (S,ADDR,SIN,SOUT) Then
    begin
    WriteLn ('Couldn''t connect to localhost');
    WriteLn ('Socket error : ',strerror(SocketError));
    halt(1);
    end;
  rewrite (sout);
  reset(sin);
  writeln (sout,paramstr(1));
  flush(sout);
  while not eof(sin) do
    begin
    readln (Sin,line);
    writeln (line);
    end;
  fpShutdown(s,2);
  close (sin);
  close (sout);
```

```
end.
```

72.4.5 fpaccept

Synopsis: Accept a connection from a socket.

Declaration: `function fpaccept(s: cint; addrx: psockaddr; addrlen: pSockLen) : cint`

Visibility: default

Description: `Accept` accepts a connection from a socket `S`, which was listening for a connection. If a connection is accepted, a file descriptor is returned (positive number). On error `-1` is returned. The returned socket may NOT be used to accept more connections. The original socket remains open.

The `Accept` call fills the address of the connecting entity in `Addrx`, and sets its length in `Addrlen`. `Addrx` should be pointing to enough space, and `Addrlen` should be set to the amount of space available, prior to the call.

Errors: On error, `-1` is returned, and errors are reported in `SocketError`, and include the following:

ESockEBADF (1263) The socket descriptor is invalid.

ESockENOTSOCK (1264) The descriptor is not a socket.

SYS_EOPNOTSUPP The socket type doesn't support the `Listen` operation.

ESockEFAULT (1263)`Addr` points outside your address space.

ESockEWOULDBLOCK (1264) The requested operation would block the process.

See also: fpListen (1299), fpConnect (1295), fpBind (1295)

Listing: ./examples/sockex/socksvr.pp

```
Program server;

{
  Program to test Sockets unit by Michael van Canneyt and Peter Vreman
  Server Version, First Run sock_svr to let it create a socket and then
  sock_cli to connect to that socket
}
{$mode fpc}
uses Sockets;

Var
  FromName : string;
  Buffer   : string[255];
  S        : Longint;
  Sin,Sout : Text;
  SAddr    : TInetSockAddr;

procedure perror (const S:string);
begin
  writeln (S,SocketError);
  halt(100);
end;

begin
  S:=fpSocket (AF_INET,SOCK_STREAM,0);
  if SocketError<>0 then
```

```
  Perror ('Server : Socket : ');
 SAddr.sin_family:=AF_INET;
 { port 50000 in network order }
 SAddr.sin_port:=htons(5000);
 SAddr.sin_addr.s_addr:=0;
 if fpBind(S,@SAddr,sizeof(saddr))=-1 then
  PError ('Server : Bind : ');
 if fpListen (S,1)=-1 then
  PError ('Server : Listen : ');
 Writeln('Waiting for Connect from Client, run now sock_cli in an other tty');
 if Accept(S,FromName,Sin,Sout) then
  PError ('Server : Accept : '+fromname);
 Reset(Sin);
 ReWrite(Sout);
 Writeln(Sout,'Message From Server');
 Flush(SOut);
 while not eof(sin) do
  begin
    Readln(Sin,Buffer);
    Writeln('Server : read : ',buffer);
  end;
end.
```

72.4.6 fpbind

Synopsis: Bind a socket to an address.

Declaration: `function fpbind(s: cint; addrx: psockaddr; addrlen: TSockLen) : cint`

Visibility: default

Description: `fpBind` binds the socket `s` to address `Addrx`. `Addrx` has length `Addrlen`. The function returns `0` if the call was successful, `-1` if not.

Errors: Errors are returned in `SocketError` and include the following:

ESockEBADF (1263)The socket descriptor is invalid.

ESockEINVAL (1263)The socket is already bound to an address,

ESockEACCESS (1263)Address is protected and you don't have permission to open it.

More errors can be found in the Unix man pages.

See also: FPSocket (1302)

72.4.7 fpconnect

Synopsis: Open a connection to a server socket.

Declaration: `function fpconnect(s: cint; name: psockaddr; namelen: TSockLen) : cint`

Visibility: default

Description: `fpConnect` uses the socket `s` to open a connection to a peer, whose address is described by `Name`. `NameLen` contains the length of the address. The type of `Name` depends on the kind of connection you are trying to make, but is generally one of `TSockAddr` or `TUnixSockAddr`.

The `fpConnect` function returns zero if the call was success full, `-1` in case of error.

Errors: On error, -1 is returned and errors are reported in SocketError.

See also: fpListen (1299), fpBind (1295), fpAccept (1294)

Listing: ./examples/sockex/sockcli.pp

```
Program Client;

{
  Program to test Sockets unit by Michael van Canneyt and Peter Vreman
  Client Version, First Run sock_svr to let it create a socket and then
  sock_cli to connect to that socket
}

uses Sockets;

procedure PError(const S : string);
begin
  writeln(S,SocketError);
  halt(100);
end;


Var
  SAddr     : TInetSockAddr;
  Buffer    : string [255];
  S         : Longint;
  Sin,Sout  : Text;
  i         : integer;

begin
  S:=fpSocket (AF_INET,SOCK_STREAM,0);
  if s=-1 then
   Perror('Client : Socket : ');
  SAddr.sin_family:=AF_INET;
  { port 50000 in network order }
  SAddr.sin_port:=htons(5000);
  { localhost : 127.0.0.1 in network order }
  SAddr.sin_addr.s_addr:=HostToNet((127 shl 24) or 1);
  if not Connect (S,SAddr,Sin,Sout) then
   PError('Client : Connect : ');
  Reset(Sin);
  ReWrite(Sout);
  Buffer:='This is a textstring sent by the Client.';
  for i:=1 to 10 do
    WriteIn(Sout,Buffer);
  Flush(Sout);
  ReadIn(SIn,Buffer);
  WriteLn(Buffer);
  Close(sout);
end.
```

Listing: ./examples/sockex/pfinger.pp

```
program pfinger;

uses sockets,errors;

Var
```

```
  Addr : TInetSockAddr;
  S : Longint;
  Sin,Sout : Text;
  Line : string;

begin
  Addr.sin_family:=AF_INET;
  { port 79 in network order }
  Addr.sin_port:=79 shl 8;
  { localhost : 127.0.0.1 in network order }
  Addr.sin_addr.s_addr:=((1 shl 24) or 127);
  S:=fpSocket(AF_INET,SOCK_STREAM,0);
  If Not Connect (S,ADDR,SIN,SOUT) Then
    begin
    Writeln ('Couldn''t connect to localhost');
    Writeln ('Socket error : ',strerror(SocketError));
    halt(1);
    end;
  rewrite (sout);
  reset(sin);
  writeln (sout,paramstr(1));
  flush(sout);
  while not eof(sin) do
    begin
    readln (Sin,line);
    writeln (line);
    end;
  fpShutdown(s,2);
  close (sin);
  close (sout);
end.
```

72.4.8 fpgetpeername

Synopsis: Return the name (address) of the connected peer.

Declaration:
```
function fpgetpeername(s: cint; name: psockaddr; namelen: pSockLen)
                              : cint
```

Visibility: default

Description: `fpGetPeerName` returns the name of the entity connected to the specified socket `S`. The Socket must be connected for this call to work.

`Name` should point to enough space to store the name, the amount of space pointed to should be set in `Namelen`. When the function returns successfully, `Name` will be filled with the name, and `Name` will be set to the length of `Name`.

Errors: Errors are reported in `SocketError`, and include the following:

ESockEBADF (1263) The socket descriptor is invalid.

ESockENOBUFS (1264) The system doesn't have enough buffers to perform the operation.

ESockENOTSOCK (1264) The descriptor is not a socket.

ESockEFAULT (1263)`Addr` points outside your address space.

ESockENOTCONN (1264) The socket isn't connected.

See also: fpConnect (1295), fpSocket (1302)

72.4.9 fpgetsockname

Synopsis: Return name of socket.

Declaration:
```
function fpgetsockname(s: cint; name: psockaddr; namelen: pSockLen)
                       : cint
```

Visibility: default

Description: `fpGetSockName` returns the current name of the specified socket `S`. `Name` should point to enough space to store the name, the amount of space pointed to should be set in `Namelen`. When the function returns successfully, `Name` will be filled with the name, and `Namelen` will be set to the length of `Name`.

Errors: Errors are reported in `SocketError`, and include the following:

ESockEBADF (1263)The socket descriptor is invalid.

ESockENOBUFS (1264)The system doesn't have enough buffers to perform the operation.

ESockENOTSOCK (1264)The descriptor is not a socket.

ESockEFAULT (1263)`Addr` points outside your address space.

See also: fpBind (1295)

72.4.10 fpgetsockopt

Synopsis: Get current socket options.

Declaration:
```
function fpgetsockopt(s: cint; level: cint; optname: cint;
                      optval: pointer; optlen: pSockLen) : cint
```

Visibility: default

Description: `fpGetSockOpt` gets the connection option `optname`, for socket `S`. The socket may be obtained from different levels, indicated by `Level`, which can be one of the following:

SOL_SOCKET From the socket itself.

XXXset `Level` to `XXX`, the protocol number of the protocol which should interpret the option.

The options are stored in the memory location pointed to by `optval`. `optlen` should point to the initial length of `optval`, and on return will contain the actual length of the stored data.

On success, 0 is returned. On Error, -1 is returned.

Errors: Errors are reported in `SocketError`, and include the following:

ESockEBADF (1263)The socket descriptor is invalid.

ESockENOTSOCK (1264)The descriptor is not a socket.

ESockEFAULT (1263)`OptVal` points outside your address space.

See also: fpSetSockOpt (1301)

72.4.11 fplisten

Synopsis: Listen for connections on a socket.

Declaration:
```
function fplisten(s: cint; backlog: cint) : cint
```

Visibility: default

Description: `fpListen` listens for up to `backlog` connections from socket `S`. The socket `S` must be of type `SOCK_STREAM` or `Sock_SEQPACKET`.

The function returns `0` if a connection was accepted, `-1` if an error occurred.

Errors: Errors are reported in `SocketError`, and include the following:

ESockEBADF (1263) The socket descriptor is invalid.

ESockENOTSOCK (1264) The descriptor is not a socket.

SYS_EOPNOTSUPP The socket type doesn't support the `Listen` operation.

See also: fpSocket (1302), fpBind (1295), fpConnect (1295)

72.4.12 fprecv

Synopsis: Receive data on socket.

Declaration:
```
function fprecv(s: cint; buf: pointer; len: size_t; flags: cint)
                      : ssize_t
```

Visibility: default

Description: `fpRecv` reads at most `len` bytes from socket `S` into address `buf`. The socket must be in a connected state. `Flags` can be one of the following:

1Process out-of band data.

4Bypass routing, use a direct interface.

??Wait for full request or report an error.

The functions returns the number of bytes actually read from the socket, or -1 if a detectable error occurred.

Errors: Errors are reported in `SocketError`, and include the following:

ESockEBADF (1263) The socket descriptor is invalid.

ESockENOTCONN (1264) The socket isn't connected.

ESockENOTSOCK (1264) The descriptor is not a socket.

ESockEFAULT (1263) The address is outside your address space.

ESockEMSGSIZE (1263) The message cannot be sent atomically.

ESockEWOULDBLOCK (1264) The requested operation would block the process.

ESockENOBUFS (1264) The system doesn't have enough free buffers available.

See also: FPSend (1300)

72.4.13 fprecvfrom

Synopsis: Receive data from an unconnected socket.

Declaration:
```
function fprecvfrom(s: cint; buf: pointer; len: size_t; flags: cint;
                    from: psockaddr; fromlen: pSockLen) : ssize_t
```

Visibility: default

Description: `fpRecvFrom` receives data in buffer `Buf` with maximum length `Len` from socket `S`. Receipt is controlled by options in `Flags`. The location pointed to by `from` will be filled with the address from the sender, and it's length will be stored in `fromlen`. The function returns the number of bytes received, or -1 on error. `AddrLen`.

Errors: On error, -1 is returned.

See also: fpSocket (1302), fprecv (1299)

72.4.14 fpsend

Synopsis: Send data through socket.

Declaration:
```
function fpsend(s: cint; msg: pointer; len: size_t; flags: cint)
                : ssize_t
```

Visibility: default

Description: `fpSend` sends `Len` bytes starting from address `Msg` to socket `S`. `S` must be in a connected state. Options can be passed in `Flags`.

The function returns the number of bytes sent, or -1 if a detectable error occurred.

`Flags` can be one of the following:

1Process out-of band data.

4Bypass routing, use a direct interface.

Errors: Errors are reported in `SocketError`, and include the following:

ESockEBADF (1263) The socket descriptor is invalid.

ESockENOTSOCK (1264) The descriptor is not a socket.

ESockEFAULT (1263) The address is outside your address space.

ESockEMSGSIZE (1263) The message cannot be sent atomically.

ESockEWOULDBLOCK (1264) The requested operation would block the process.

ESockENOBUFS (1264) The system doesn't have enough free buffers available.

See also: fpRecv (1299)

72.4.15 fpsendto

Synopsis: Send data through an unconnected socket to an address.

Declaration:
```
function fpsendto(s: cint; msg: pointer; len: size_t; flags: cint;
                  tox: psockaddr; tolen: TSockLen) : ssize_t
```

Visibility: default

Description: `fpSendTo` sends data from buffer `Msg` with length `len` through socket `S` with options `Flags`. The data is sent to address `tox`, which has length `toLen`

Errors: On error, -1 is returned.

See also: fpSocket (1302), fpSend (1300), fpRecvFrom (1300)

72.4.16 fpsetsockopt

Synopsis: Set socket options.

Declaration:
```
function fpsetsockopt(s: cint; level: cint; optname: cint;
                      optval: pointer; optlen: TSockLen) : cint
```

Visibility: default

Description: `fpSetSockOpt` sets the connection options for socket `S`. The socket may be manipulated at different levels, indicated by `Level`, which can be one of the following:

SOL_SOCKET To manipulate the socket itself.

XXX set `Level` to `XXX`, the protocol number of the protocol which should interpret the option.

The actual option is stored in a buffer pointed to by `optval`, with length `optlen`.

For more information on this call, refer to the UNIX manual page setsockopt

Errors: Errors are reported in `SocketError`, and include the following:

ESockEBADF (1263) The socket descriptor is invalid.

ESockENOTSOCK (1264) The descriptor is not a socket.

ESockEFAULT (1263) `OptVal` points outside your address space.

See also: fpGetSockOpt (1298)

72.4.17 fpshutdown

Synopsis: Close one end of full duplex connection.

Declaration:
```
function fpshutdown(s: cint; how: cint) : cint
```

Visibility: default

Description: `fpShutDown` closes one end of a full duplex socket connection, described by `S`. The parameter `How` determines how the connection will be shut down, and can be one of the following:

0 Further receives are disallowed.

1 Further sends are disallowed.

2 Sending nor receiving are allowed.

On success, the function returns 0, on error -1 is returned.

Errors: `SocketError` is used to report errors, and includes the following:

ESockEBADF (1263) The socket descriptor is invalid.

ESockENOTCONN (1264) The socket isn't connected.

ESockENOTSOCK (1264) The descriptor is not a socket.

See also: fpSocket (1302), fpConnect (1295)

72.4.18 fpsocket

Synopsis: Create new socket.

Declaration:
```
function fpsocket(domain: cint; xtype: cint; protocol: cint) : cint
```

Visibility: default

Description: `fpSocket` creates a new socket in domain `Domain`, from type `xType` using protocol `Protocol`. The Domain, Socket type and Protocol can be specified using predefined constants (see the section on constants for available constants) If successful, the function returns a socket descriptor, which can be passed to a subsequent fpBind (1295) call. If unsuccessfully, the function returns -1.

for an example, see Accept (1290).

Errors: Errors are returned in `SocketError`, and include the following:

ESockEPROTONOSUPPORT (1264) The protocol type or the specified protocol is not supported within this domain.

ESockEMFILE (1263) The per-process descriptor table is full.

SYS_ENFILE The system file table is full.

ESockEACCESS (1263) Permission to create a socket of the specified type and/or protocol is denied.

ESockENOBUFS (1264) Insufficient buffer space is available. The socket cannot be created until sufficient resources are freed.

See also: FPSocketPair (1302)

72.4.19 fpsocketpair

Synopsis: Create socket pair.

Declaration:
```
function fpsocketpair(d: cint; xtype: cint; protocol: cint; sv: pcint)
                      : cint
```

Visibility: default

Description: `fpSocketPair` creates 2 sockets in domain `D`, from type `xType` and using protocol `Protocol`. The pair is returned in `sv`, and they are indistinguishable. The function returns -1 upon error and 0 upon success.

Errors: Errors are reported in `SocketError`, and are the same as in FPSocket (1302)

See also: Str2UnixSockAddr (1306)

72.4.20 HostAddrToStr

Synopsis: Convert a host address to a string.

Declaration:
```
function HostAddrToStr(Entry: in_addr) : AnsiString
```

Visibility: default

Description: `HostAddrToStr` converts the host address in `Entry` to a string representation in human-readable form (a dotted quad).

Basically, it is the same as NetAddrToStr (1304), but with the bytes in correct order.

See also: NetAddrToStr (1304), StrToHostAddr (1306), StrToNetAddr (1307)

72.4.21 HostAddrToStr6

Synopsis: Convert a IPV6 host address to a string representation.

Declaration:
```
function HostAddrToStr6(Entry: Tin6_addr) : AnsiString
```

Visibility: default

Description: `HostAddrToStr6` converts the IPV6 host address in `Entry` to a string representation in human-readable form.

Basically, it is the same as NetAddrToStr6 (1304), but with the bytes in correct order.

See also: NetAddrToStr (1304), StrToHostAddr (1306), StrToNetAddr (1307), StrToHostAddr6 (1307)

72.4.22 HostToNet

Synopsis: Convert a host address to a network address.

Declaration:
```
function HostToNet(Host: in_addr) : in_addr
function HostToNet(Host: LongInt) : LongInt
```

Visibility: default

Description: `HostToNet` converts a host address to a network address. It takes care of endianness of the host machine. The address can be specified as a dotted quad or as a longint.

Errors: None.

See also: NetToHost (1304), NToHS (1305), HToNS (1303), ShortHostToNet (1305), ShortNetToHost (1305)

72.4.23 htonl

Synopsis: Convert long integer from host ordered to network ordered.

Declaration:
```
function htonl(host: Cardinal) : Cardinal;  Overload
```

Visibility: default

Description: `htonl` makes sure that the bytes in `host` are ordered in the correct way for sending over the network and returns the correctly ordered result.

See also: htons (1303), ntohl (1304), ntohs (1305)

72.4.24 htons

Synopsis: Convert short integer from host ordered to network ordered.

Declaration:
```
function htons(host: Word) : Word
```

Visibility: default

Description: `htons` makes sure that the bytes in `host` are ordered in the correct way for sending over the network and returns the correctly ordered result.

See also: htonl (1303), ntohl (1304), ntohs (1305)

72.4.25 NetAddrToStr

Synopsis: Convert a network address to a string.

Declaration:
```
function NetAddrToStr(Entry: in_addr) : AnsiString
```

Visibility: default

Description: `NetAddrToStr` converts the network address in `Entry` to a string representation in human-readable form (a dotted quad).

See also: HostAddrToStr (1302), StrToNetAddr (1307), StrToHostAddr (1306)

72.4.26 NetAddrToStr6

Synopsis: Convert a IPV6 network address to a string.

Declaration:
```
function NetAddrToStr6(Entry: Tin6_addr) : AnsiString
```

Visibility: default

Description: `NetAddrToStr6` converts the IPV6 network address in `Entry` to a string representation in human-readable form.

Basically, it is the same as NetAddrToStr6 (1304), but with the bytes in correct order.

See also: NetAddrToStr (1304), StrToHostAddr (1306), StrToNetAddr (1307), StrToHostAddr6 (1307)

72.4.27 NetToHost

Synopsis: Convert a network address to a host address.

Declaration:
```
function NetToHost(Net: in_addr) : in_addr
function NetToHost(Net: LongInt) : LongInt
```

Visibility: default

Description: `NetToHost` converts a network address to a host address. It takes care of endianness of the host machine. The address can be specified as a dotted quad or as a longint.

Errors: None.

See also: HostToNet (1303), NToHS (1305), HToNS (1303), ShortHostToNet (1305), ShortNetToHost (1305)

72.4.28 NToHl

Synopsis: Convert long integer from network ordered to host ordered.

Declaration:
```
function NToHl(Net: Cardinal) : Cardinal;  Overload
```

Visibility: default

Description: `ntohs` makes sure that the bytes in `Net`, received from the network, are ordered in the correct way for handling by the host machine, and returns the correctly ordered result.

See also: htonl (1303), htons (1303), ntohs (1305)

72.4.29 NToHs

Synopsis: Convert short integer from network ordered to host ordered.

Declaration: `function NToHs(Net: Word) : Word`

Visibility: default

Description: `ntohs` makes sure that the bytes in `Net`, received from the network, are ordered in the correct way for handling by the host machine, and returns the correctly ordered result.

See also: htonl (1303), htons (1303), ntohl (1304)

72.4.30 ShortHostToNet

Synopsis: Convert a host port number to a network port number.

Declaration: `function ShortHostToNet(Host: Word) : Word`

Visibility: default

Description: `ShortHostToNet` converts a host port number to a network port number. It takes care of endianness of the host machine.

Errors: None.

See also: ShortNetToHost (1305), HostToNet (1303), NToHS (1305), HToNS (1303)

72.4.31 ShortNetToHost

Synopsis: Convert a network port number to a host port number.

Declaration: `function ShortNetToHost(Net: Word) : Word`

Visibility: default

Description: `ShortNetToHost` converts a network port number to a host port number. It takes care of endianness of the host machine.

Errors: None.

See also: ShortNetToHost (1305), HostToNet (1303), NToHS (1305), HToNS (1303)

72.4.32 Sock2File

Synopsis: Convert socket to untyped file descriptors.

Declaration: `procedure Sock2File(Sock: LongInt; var SockIn: File; var SockOut: File)`

Visibility: default

Description: `Sock2File` transforms a socket `Sock` into 2 Pascal file descriptors of type `File`, one for reading from the socket (`SockIn`), one for writing to the socket (`SockOut`).

Errors: None.

See also: FPSocket (1302), Sock2Text (1306)

72.4.33 Sock2Text

Synopsis: Convert socket to text file descriptors.

Declaration:
```
procedure Sock2Text(Sock: LongInt; var SockIn: Text; var SockOut: Text)
```

Visibility: default

Description: `Sock2Text` transforms a socket `Sock` into 2 Pascal file descriptors of type `Text`, one for reading from the socket (`SockIn`), one for writing to the socket (`SockOut`).

Errors: None.

See also: FPSocket (1302), Sock2File (1305)

72.4.34 socketerror

Synopsis: Contains the error code for the last socket operation.

Declaration:
```
function socketerror : cint
```

Visibility: default

Description: `SocketError` contains the error code for the last socket operation. It can be examined to return the last socket error.

72.4.35 Str2UnixSockAddr

Synopsis: Convert path to TUnixSockAddr (1309).

Declaration:
```
procedure Str2UnixSockAddr(const addr: string; var t: TUnixSockAddr;
                           var len: LongInt)
```

Visibility: default

Description: `Str2UnixSockAddr` transforms a Unix socket address in a string to a `TUnixSockAddr` structure which can be passed to the Bind (1291) call.

Errors: None.

See also: FPSocket (1302), FPBind (1295)

72.4.36 StrToHostAddr

Synopsis: Convert a string to a host address.

Declaration:
```
function StrToHostAddr(IP: AnsiString) : in_addr
```

Visibility: default

Description: `StrToHostAddr` converts the string representation in `IP` to a host address and returns the host address.

Errors: On error, the host address is filled with zeroes.

See also: NetAddrToStr (1304), HostAddrToStr (1302), StrToNetAddr (1307)

72.4.37 StrToHostAddr6

Synopsis: Convert a string to a IPV6 host address.

Declaration: `function StrToHostAddr6(IP: AnsiString) : Tin6_addr`

Visibility: default

Description: `StrToHostAddr6` converts the string representation in `IP` to a IPV6 host address and returns the host address.

Errors: On error, the address is filled with zeroes.

See also: NetAddrToStr6 (1304), HostAddrToStr6 (1303), StrToHostAddr (1306)

72.4.38 StrToNetAddr

Synopsis: Convert a string to a network address.

Declaration: `function StrToNetAddr(IP: AnsiString) : in_addr`

Visibility: default

Description: `StrToNetAddr` converts the string representation in `IP` to a network address and returns the network address.

Errors: On error, the network address is filled with zeroes.

See also: NetAddrToStr (1304), HostAddrToStr (1302), StrToHostAddr (1306)

72.4.39 StrToNetAddr6

Synopsis: Convert a string to a IPV6 network address.

Declaration: `function StrToNetAddr6(IP: AnsiString) : Tin6_addr`

Visibility: default

Description: `StrToNetAddr6` converts the string representation in `IP` to a IPV6 network address and returns the network address.

Errors: On error, the address is filled with zeroes.

See also: NetAddrToStr6 (1304), HostAddrToStr6 (1303), StrToHostAddr6 (1307)

72.4.40 TryStrToHostAddr

Synopsis: Try to convert a string to an IPV4 address.

Declaration: `function TryStrToHostAddr(IP: AnsiString; out ip4: in_addr) : Boolean`

Visibility: default

Description: `TryStrToHostAddr` attempts to convert `IP` to an IPV4 address. If successful it returns the address in `ip4` and returns `True`. If unsuccessful it returns `False`.

See also: StrToHostAddr (1306), TryStrToHostAddr6 (1308)

72.4.41 TryStrToHostAddr6

Synopsis: Try to convert a string to an IPV6 address.

Declaration: `function TryStrToHostAddr6(IP: AnsiString; out ip6: in6_addr) : Boolean`

Visibility: default

Description: `TryStrToHostAddr` attempts to convert `IP` to an IPV6 address. If successful it returns the address in `ip6` and returns `True`. If unsuccessful it returns `False`.

See also: StrToHostAddr6 (1307), TryStrToHostAddr (1307)

72.5 linger

```
linger = packed record
  l_onoff : cint;
  l_linger : cint;
end
```

This record is used in the fpsetsockopt (1301) call to specify linger options.

72.6 sockaddr_in

```
sockaddr_in = packed record
  sin_family : sa_family_t;
  sin_port
  : cushort;
  sin_addr : in_addr;
  xpad : Array[0..7] of Char;
end
```

`sockaddr_in` is used to store a INET socket address for the FPBind (1295), FPRecv (1299) and FPSend (1300) calls.

72.7 sockaddr_in6

```
sockaddr_in6 = packed record
  sin6_family : sa_family_t;
  sin6_port
  : cuint16;
  sin6_flowinfo : cuint32;
  sin6_addr : in6_addr;
  sin6_scope_id
  : cuint32;
end
```

Alias for sockaddr_in6 (1308).

72.8 sockaddr_un

```
sockaddr_un = packed record
  sun_family : sa_family_t;
  sun_path
  : Array[0..107] of Char;
end
```

sockaddr_un is used to store a UNIX socket address for the FPBind (1295), FPRecv (1299) and FPSend (1300) calls.

72.9 TUnixSockAddr

```
TUnixSockAddr = packed record
  family : sa_family_t;
  path : Array
  [0..107] of Char;
end
```

Alias for sockaddr_un (1309).

72.10 ucred

```
ucred = record
  pid : cuint32;
  uid : cuint32;
  gid : cuint32;
  end
```

ucred is used for socket control messages with type SCM_CREDENTIALS.

pid Process ID.

uid User ID.

gid Group ID.

Chapter 73

Reference for unit 'Strings'

73.1 Used units

Table 73.1: Used units by unit 'Strings'

Name	Page
System	1362

73.2 Overview

This chapter describes the `STRINGS` unit for Free Pascal. This unit is system independent, and therefore works on all supported platforms.

73.3 Procedures and functions

73.3.1 stralloc

Synopsis: Allocate memory for a new null-terminated string on the heap.

Declaration: `function stralloc(L: SizeInt) : PChar`

Visibility: default

Description: `StrAlloc` reserves memory on the heap for a string with length `Len`, terminating `#0` included, and returns a pointer to it.

Errors: If there is not enough memory, a run-time error occurs.

See also: StrNew (1319), StrPCopy (1320)

73.3.2 strcat

Synopsis: Concatenate 2 null-terminated strings.

Declaration: `function strcat(dest: PChar; source: PChar) : PChar`

Visibility: default

Description: Attaches `Source` to `Dest` and returns `Dest`.

Errors: No length checking is performed.

See also: StrLCat (1315)

Listing: ./examples/stringex/ex11.pp

```
Program Example11;

Uses strings;

{ Program to demonstrate the StrCat function. }

Const P1 : PChar = 'This is a PChar String.';

Var P2 : PChar;

begin
  P2:=StrAlloc (StrLen(P1)*2+1);
  StrMove (P2,P1,StrLen(P1)+1); { P2=P1 }
  StrCat (P2,P1);               { Append P2 once more }
  Writeln ('P2 : ',P2);
  StrDispose(P2);
end.
```

73.3.3 strcomp

Synopsis: Compare 2 null-terminated strings, case sensitive.

Declaration: `function strcomp(str1: PChar; str2: PChar) : SizeInt`

Visibility: default

Description: Compares the null-terminated strings `S1` and `S2`. The result is

- A negative `SizeInt` when `S1<S2`.
- 0 when `S1=S2`.
- A positive `SizeInt` when `S1>S2`.

For an example, see StrLComp (1316).

Errors: None.

See also: StrLComp (1316), StrIComp (1314), StrLIComp (1317)

73.3.4 strcopy

Synopsis: Copy a null-terminated string.

Declaration: `function strcopy(dest: PChar; source: PChar) : PChar;  Overload`

Visibility: default

Description: Copy the null terminated string in `Source` to `Dest`, and returns a pointer to `Dest`. `Dest` needs enough room to contain `Source`, i.e. `StrLen(Source)+1` bytes.

Errors: No length checking is performed.

See also: StrPCopy (1320), StrLCopy (1316), StrECopy (1313)

Listing: ./examples/stringex/ex4.pp

```
Program Example4;

Uses strings;

{ Program to demonstrate the StrCopy function. }

Const P : PCHar = 'This is a PCHAR string.';

var PP : PChar;

begin
  PP:=StrAlloc(Strlen(P)+1);
  STrCopy (PP,P);
  If StrComp (PP,P)<>0 then
    WriteIn ('Oh-oh problems...')
  else
    WriteIn ('All is well : PP=',PP);
  StrDispose(PP);
end.
```

73.3.5 strdispose

Synopsis: disposes of a null-terminated string on the heap.

Declaration: `procedure strdispose(p: PChar)`

Visibility: default

Description: Removes the string in `P` from the heap and releases the memory.

Errors: None.

See also: StrNew (1319)

Listing: ./examples/stringex/ex17.pp

```
Program Example17;

Uses strings;

{ Program to demonstrate the StrDispose function. }

Const P1 : PChar = 'This is a PChar string';

var P2 : PChar;

begin
  P2:=StrNew (P1);
  WriteIn ('P2 : ',P2);
  StrDispose(P2);
end.
```

73.3.6 strecopy

Synopsis: Copy a null-terminated string, return a pointer to the end.

Declaration: `function strecopy(dest: PChar; source: PChar) : PChar`

Visibility: default

Description: Copies the Null-terminated string in `Source` to `Dest`, and returns a pointer to the end (i.e. the terminating Null-character) of the copied string.

Errors: No length checking is performed.

See also: StrLCopy (1316), StrCopy (1311)

Listing: ./examples/stringex/ex6.pp

```
Program Example6;

Uses strings;

{ Program to demonstrate the StrECopy function. }

Const P : PChar = 'This is a PCHAR string.';

Var PP : PChar;

begin
  PP:=StrAlloc (StrLen(P)+1);
  If Longint(StrECopy(PP,P))-Longint(PP)<>StrLen(P) then
    WriteIn('Something is wrong here !')
  else
    WriteIn ('PP= ',PP);
  StrDispose(PP);
end.
```

73.3.7 strend

Synopsis: Return a pointer to the end of a null-terminated string.

Declaration: `function strend(p: PChar) : PChar`

Visibility: default

Description: Returns a pointer to the end of `P`. (i.e. to the terminating null-character.

Errors: None.

See also: StrLen (1317)

Listing: ./examples/stringex/ex7.pp

```
Program Example6;

Uses strings;

{ Program to demonstrate the StrEnd function. }

Const P : PChar = 'This is a PCHAR string.';
```

```
begin
  If Longint(StrEnd(P))-Longint(P)<>StrLen(P) then
    WriteIn('Something is wrong here !')
  else
    WriteIn ('All is well..');
end.
```

73.3.8 stricomp

Synopsis: Compare 2 null-terminated strings, case insensitive.

Declaration: `function stricomp(str1: PChar; str2: PChar) : SizeInt`

Visibility: default

Description: Compares the null-terminated strings `S1` and `S2`, ignoring case. The result is

- A negative `SizeInt` when `S1<S2`.
- 0 when `S1=S2`.
- A positive `SizeInt` when `S1>S2`.

Errors: None.

See also: StrLComp (1316), StrComp (1311), StrLIComp (1317)

Listing: ./examples/stringex/ex8.pp

```
Program Example8;

Uses strings;

{ Program to demonstrate the StrLComp function. }

Const P1 : PChar = 'This is the first string.';
      P2 : PCHar = 'This is the second string.';

Var L : Longint;

begin
  Write ('P1 and P2 are ');
  If StrComp (P1,P2)<>0 then write ('NOT ');
  write ('equal. The first ');
  L:=1;
  While StrLComp(P1,P2,L)=0 do inc (L);
  dec(l);
  WriteIn (l,' characters are the same.');
end.
```

73.3.9 stripos

Synopsis: Return the position of a substring in a string, case insensitive.

Declaration: `function stripos(str1: PChar; str2: PChar) : PChar`

Visibility: default

Description: `stripos` returns the position of `str2` in `str1`. It searches in a case-insensitive manner, and if it finds a match, it returns a pointer to the location of the match. If no match is found, `Nil` is returned.

Errors: No checks are done on the validity of the pointers, and the pointers are assumed to point to a properly null-terminated string. If either of these conditions are not met, a run-time error may follow.

See also: striscan (1315), strpos (1321)

73.3.10 striscan

Synopsis: Scan a string for a character, case-insensitive.

Declaration: `function striscan(p: PChar; c: Char) : PChar`

Visibility: default

Description: `striscan` does the same as strscan (1322) but compares the characters case-insensitively. It returns a pointer to the first occurrence of the character `c` in the null-terminated string `p`, or `Nil` if `c` is not present in the string.

See also: strscan (1322), strriscan (1321)

73.3.11 strlcat

Synopsis: Concatenate 2 null-terminated strings, with length boundary.

Declaration: `function strlcat(dest: PChar; source: PChar; l: SizeInt) : PChar`

Visibility: default

Description: Adds `L` characters from `Source` to `Dest`, and adds a terminating null-character. Returns `Dest`.

Errors: None.

See also: StrCat (1310)

Listing: ./examples/stringex/ex12.pp

```
Program Example12;

Uses strings;

{ Program to demonstrate the StrLCat function. }

Const P1 : PChar = '1234567890';

Var P2 : PChar;

begin
  P2:=StrAlloc (StrLen(P1)*2+1);
  P2^:=#0; { Zero length }
  StrCat (P2,P1);
  StrLCat (P2,P1,5);
  WriteIn ('P2 = ',P2);
  StrDispose(P2)
end.
```

73.3.12 strlcomp

Synopsis: Compare limited number of characters of 2 null-terminated strings.

Declaration:
```
function strlcomp(str1: PChar; str2: PChar; l: SizeInt) : SizeInt
```

Visibility: default

Description: Compares maximum `L` characters of the null-terminated strings `S1` and `S2`. The result is

- A negative `SizeInt` when `S1<S2`.
- 0 when `S1=S2`.
- A positive `SizeInt` when `S1>S2`.

Errors: None.

See also: StrComp (1311), StrIComp (1314), StrLIComp (1317)

Listing: ./examples/stringex/ex8.pp

```
Program Example8;

Uses strings;

{ Program to demonstrate the StrLComp function. }

Const P1 : PChar = 'This is the first string.';
      P2 : PCHar = 'This is the second string.';

Var L : Longint;

begin
  Write ('P1 and P2 are ');
  If StrComp (P1,P2)<>0 then write ('NOT ');
  write ('equal. The first ');
  L:=1;
  While StrLComp(P1,P2,L)=0 do inc (L);
  dec(l);
  Writeln (l,' characters are the same.');
end.
```

73.3.13 strlcopy

Synopsis: Copy a null-terminated string, limited in length.

Declaration:
```
function strlcopy(dest: PChar; source: PChar; maxlen: SizeInt) : PChar
                  ;  Overload
```

Visibility: default

Description: Copies `MaxLen` characters from `Source` to `Dest`, and makes `Dest` a null terminated string.

Errors: No length checking is performed.

See also: StrCopy (1311), StrECopy (1313)

Listing: ./examples/stringex/ex5.pp

```
Program Example5;

Uses strings;

{ Program to demonstrate the StrLCopy function. }

Const P : PCHar = '123456789ABCDEF';

var PP : PCHar;

begin
  PP:=StrAlloc(11);
  WriteIn ('First 10 characters of P : ',StrLCopy (PP,P,10));
  StrDispose(PP);
end.
```

73.3.14 strlen

Synopsis: Length of a null-terminated string.

Declaration: `function strlen(p: PChar) : SizeInt`

Visibility: default

Description: Returns the length of the null-terminated string `P`. If `P` equals `Nil`, then zero (0) is returned.

Errors: None.

See also: StrNew (1319)

Listing: ./examples/stringex/ex1.pp

```
Program Example1;

Uses strings;

{ Program to demonstrate the StrLen function. }

Const P : PChar = 'This is a constant pchar string';

begin
  WriteIn ('P         : ',p);
  WriteIn ('length(P) : ',StrLen(P));
end.
```

73.3.15 strlicomp

Synopsis: Compare limited number of characters in 2 null-terminated strings, ignoring case.

Declaration: `function strlicomp(str1: PChar; str2: PChar; l: SizeInt) : SizeInt`

Visibility: default

Description: Compares maximum `L` characters of the null-terminated strings `S1` and `S2`, ignoring case. The result is

- A negative `SizeInt` when `S1<S2`.
- 0 when `S1=S2`.
- A positive `SizeInt` when `S1>S2`.

For an example, see StrIComp (1314)

Errors: None.

See also: StrLComp (1316), StrComp (1311), StrIComp (1314)

73.3.16 strlower

Synopsis: Convert null-terminated string to all-lowercase.

Declaration: `function strlower(p: PChar) : PChar`

Visibility: default

Description: Converts `P` to an all-lowercase string. Returns `P`.

Errors: None.

See also: StrUpper (1322)

Listing: ./examples/stringex/ex14.pp

```
Program Example14;

Uses strings;

{ Program to demonstrate the StrLower and StrUpper functions. }

Const
    P1 : PChar = 'THIS IS AN UPPERCASE PCHAR STRING';
    P2 : PChar = 'this is a lowercase string';


begin
  P1:=StrNew(P1);
  P2:=strNew(P2);
  WriteIn ('Uppercase : ',StrUpper(P2));
  StrLower(P1);
  WriteIn ('Lowercase : ',P1);
  StrDispose(P1);
  StrDispose(P2);
end.
```

73.3.17 strmove

Synopsis: Move a null-terminated string to new location.

Declaration: `function strmove(dest: PChar; source: PChar; l: SizeInt) : PChar`

Visibility: default

Description: Copies `MaxLen` characters from `Source` to `Dest`. No terminating null-character is copied. Returns `Dest`

Errors: None.

See also: StrLCopy (1316), StrCopy (1311)

Listing: ./examples/stringex/ex10.pp

```
Program Example10;

Uses strings;

{ Program to demonstrate the StrMove function. }

Const P1 : PCHAR = 'This is a pchar string.';


Var P2 : Pchar;

begin
  P2:=StrAlloc(StrLen(P1)+1);
  StrMove (P2,P1,StrLen(P1)+1); { P2:=P1 }
  Writeln ('P2 = ',P2);
  StrDispose(P2);
end.
```

73.3.18 strnew

Synopsis: Allocate room for new null-terminated string.

Declaration: `function strnew(p: PChar) : PChar`

Visibility: default

Description: Copies `P` to the Heap, and returns a pointer to the copy.

Errors: Returns `Nil` if no memory was available for the copy.

See also: StrCopy (1311), StrDispose (1312)

Listing: ./examples/stringex/ex16.pp

```
Program Example16;

Uses strings;

{ Program to demonstrate the StrNew function. }

Const P1 : PChar = 'This is a PChar string';

var P2 : PChar;

begin
  P2:=StrNew (P1);
  If P1=P2 then
    writeln ('This can''t be happening...')
  else
    writeln ('P2 : ',P2);
  StrDispose(P2);
end.
```

73.3.19 strpas

Synopsis: Convert a null-terminated string to a shortstring.

Declaration: `function strpas(p: PChar) : shortstring`

Visibility: default

Description: Converts a null terminated string in `P` to a Pascal string, and returns this string. The string is truncated at 255 characters.

Errors: None.

See also: StrPCopy (1320)

Listing: ./examples/stringex/ex3.pp

```
Program Example3;

Uses strings;

{ Program to demonstrate the StrPas function. }

Const P : PChar = 'This is a PCHAR string';

var S : string;

begin
  S:=StrPas (P);
  Writeln ('S : ',S);
end.
```

73.3.20 strpcopy

Synopsis: Copy a pascal string to a null-terminated string.

Declaration: `function strpcopy(d: PChar; const s: string) : PChar`

Visibility: default

Description: Converts the Pascal string in `Se` to a Null-terminated string, and copies it to `D`. `D` needs enough room to contain the string `Source`, i.e. `Length(S)+1` bytes.

Errors: No length checking is performed.

See also: StrPas (1320)

Listing: ./examples/stringex/ex2.pp

```
Program Example2;

Uses strings;

{ Program to demonstrate the StrPCopy function. }

Const S = 'This is a normal string.';

Var P : Pchar;
```

```
begin
  p:=StrAlloc (length(S)+1);
  if StrPCopy (P,S)<>P then
    WriteIn ('This is impossible !!')
  else
    writeln (P);
  StrDispose(P);
end.
```

73.3.21 strpos

Synopsis: Search for a null-terminated substring in a null-terminated string.

Declaration: `function strpos(str1: PChar; str2: PChar) : PChar`

Visibility: default

Description: Returns a pointer to the first occurrence of `S2` in `S1`. If `S2` does not occur in `S1`, returns `Nil`.

Errors: None.

See also: StrScan (1322), StrRScan (1322)

Listing: ./examples/stringex/ex15.pp

```
Program Example15;

Uses strings;

{ Program to demonstrate the StrPos function. }

Const P : PChar = 'This is a PChar string.';
      S : Pchar = 'is';
begin
  WriteIn ('Position of ''is'' in P : ',sizeint(StrPos(P,S))-sizeint(P));
end.
```

73.3.22 strriscan

Synopsis: Scan a string reversely for a character, case-insensitive.

Declaration: `function strriscan(p: PChar; c: Char) : PChar`

Visibility: default

Description: `strriscan` does the same as strrscan (1322) but compares the characters case-insensitively. It returns a pointer to the last occurrence of the character `c` in the null-terminated string `p`, or `Nil` if `c` is not present in the string.

See also: strrscan (1322), striscan (1315)

73.3.23 strrscan

Synopsis: Find last occurrence of a character in a null-terminated string.

Declaration: `function strrscan(p: PChar; c: Char) : PChar`

Visibility: default

Description: Returns a pointer to the last occurrence of the character `C` in the null-terminated string `P`. If `C` does not occur, returns `Nil`.

For an example, see StrScan (1322).

Errors: None.

See also: StrScan (1322), StrPos (1321)

73.3.24 strscan

Synopsis: Find first occurrence of a character in a null-terminated string.

Declaration: `function strscan(p: PChar; c: Char) : PChar`

Visibility: default

Description: Returns a pointer to the first occurrence of the character `C` in the null-terminated string `P`. If `C` does not occur, returns `Nil`.

Errors: None.

See also: StrRScan (1322), StrPos (1321)

Listing: ./examples/stringex/ex13.pp

```
Program Example13;

Uses strings;

{ Program to demonstrate the StrScan and StrRScan functions. }

Const P : PChar = 'This is a PCHAR string.';
      S : Char = 's' ;

begin
  Writeln ('P, starting from first ''s'' : ',StrScan(P,s));
  Writeln ('P, starting from last ''s'' : ',StrRScan(P,s));
end.
```

73.3.25 strupper

Synopsis: Convert null-terminated string to all-uppercase.

Declaration: `function strupper(p: PChar) : PChar`

Visibility: default

Description: Converts `P` to an all-uppercase string. Returns `P`.

For an example, see StrLower (1318)

Errors: None.

See also: StrLower (1318)

Chapter 74

Reference for unit 'StrUtils'

74.1 Used units

Table 74.1: Used units by unit 'StrUtils'

Name	Page
System	1362
sysutils	1635
Types	1965

74.2 Constants, types and variables

74.2.1 Resource strings

```
SErrAmountStrings =
  'Amount of search and replace strings don''t match'
```

Error message used in stringsreplace function.

```
SInvalidRomanNumeral = '%s is not a valid Roman numeral'
```

Error string shown in exception raised when invalid roman numeral is encountered.

74.2.2 Constants

```
AnsiResemblesProc : TCompareTextProc = @ SoundexProc
```

This procedural variable is standard set to SoundexProc (1354) but can be overriden with a user-defined algorithm. This algorithm should return `True` if `AText` resembles `AOtherText`, or `False` otherwise. The standard routine compares the soundexes of the two strings and returns `True` if they are equal.

```
Brackets = ['(', ')', '[', ']', '{', '}']
```

Set of characters that contain all possible bracket characters.

```
DigitChars = ['0'..'9']
```

Set of digit characters.

```
ResemblesProc : TCompareTextProc = @ SoundexProc
```

`ResemblesProc` is a procedure variable which can be set to compare 2 string. It is similar to AnsiResemblesProc (1323), and by default is set to the SoundExProc (1354) function.

```
StdSwitchChars = ['-', '/']
```

Standard characters for the `SwitchChars` argument of GetCmdLineArg (1339).

```
StdWordDelims = [#0..' ', ',', '.', ';', '/', '\', ':', '''', '"'
  , '`'] + Brackets
```

Standard word delimiter values.

```
WordDelimiters : Set of Char = [#0..#255] - ['a'..'z', 'A'..'Z', '1'
  ..'9', '0']
```

Standard word delimiters, used in the SearchBuf (1353) call.

74.2.3 Types

```
SizeIntArray = Array of SizeInt
```

`SizeIntArray` is an auxiliary type used in StringReplace (1357)

```
TCompareTextProc = function(const AText: string; const AOther: string
  )
                                : Boolean
```

Function prototype for comparing two string in AnsiResemblesText (1330).

```
TRawByteStringArray = Array of RawByteString
```

`TRawByteStringArray` defines a dynamic array of `RawByteString`. It is used in the `RawByteString` version of the SplitCommandLine (1355) function.

```
TRomanConversionStrictness = (rcsStrict,rcsRelaxed,rcsDontCare)
```

Table 74.2: Enumeration values for type TRomanConversionStrictness

Value	Explanation
rcsDontCare	Do not check correctness.
rcsRelaxed	Like `rcsStrict` but allow more than 3 consecutive identical letters.
rcsStrict	Only accept correct roman numerals.

`TRomanConversionStrictness` is an enumerated type that can be used to decide how to react to invalid roman numerals.

rcsStrict Strict adherence to roman numerals. Up to 3 consecutive identical letters. No negative numbers. Ordering must be correct.

rcsRelaxed Same as `rcsStrict` but allow more than 3 consecutive identical letters.

rcsDontCare Do not check validity at all

```
TSoundexIntLength = 1..8
```

Range of allowed integer soundex lengths.

```
TSoundexLength = 1..MaxInt
```

Range of allowed soundex lengths.

```
TStringReplaceAlgorithm = (sraDefault,sraManySmall,sraBoyerMoore)
```

Table 74.3: Enumeration values for type TStringReplaceAlgorithm

Value	Explanation
sraBoyerMoore	Use a Boyer-Moore search algorithm.
sraDefault	Use the sysutils algorithm, which does a straightforward linear search and replace.
sraManySmall	Use an approach which is suitable for a string with many occurrences of the same small text.

`TStringReplaceAlgorithm` enumerates the available algorithms to StringReplace (1357)

sraDefault Use the sysutils algorithm, which does a straightforward linear search and replace.

sraManySmall Use an approach which is suitable for a string with many occurrences of the same small text.

sraBoyerMoore Use a Boyer-Moore search algorithm.

Depending on the kind of data that is being treated, one or the other of these algorithms may produce faster results.

```
TStringSeachOption = TStringSearchOption
```

There is an typo error in the original Borland StrUtils unit. This type just refers to the correct TStringSearchOption (1325) and is provided for compatibility only.

```
TStringSearchOption = (soDown,soMatchCase,soWholeWord)
```

Table 74.4: Enumeration values for type TStringSearchOption

Value	Explanation
soDown	Search in down direction.
soMatchCase	Match case.
soWholeWord	Search whole words only.

Possible options for SearchBuf (1353) call.

```
TStringSearchOptions = Set of TStringSearchOption
```

Set of options for SearchBuf (1353) call.

```
TUnicodeStringArray = Array of UnicodeString
```

`TUnicodeStringArray` defines a dynamic array of `UnicodeString`. It is used in the `UnicodeString` version of the SplitCommandLine (1355) function.

74.3 Procedures and functions

74.3.1 AddChar

Synopsis: Add characters to the left of a string till a certain length.

Declaration:
```
function AddChar(C: Char; const S: string; N: Integer) : string
```

Visibility: default

Description: `AddChar` adds characters (`C`) to the left of S till the length `N` is reached, and returns the resulting string. If the length of `S` is already equal to or larger than `N`, then no characters are added. The resulting string can be thought of as a right-aligned version of `S`, with length `N`.

Errors: None

See also: AddCharR (1326), PadLeft (1347), PadRight (1347), PadCenter (1347)

74.3.2 AddCharR

Synopsis: Add chars at the end of a string till it reaches a certain length.

Declaration:
```
function AddCharR(C: Char; const S: string; N: Integer) : string
```

Visibility: default

Description: `AddCharR` adds characters (`C`) to the right of S till the length `N` is reached, and returns the resulting string. If the length of `S` is already equal to or larger than `N`, then no characters are added. The resulting string can be thought of as a left-aligned version of `S`, with length `N` .

Errors: None

See also: AddChar (1326)

74.3.3 AnsiContainsStr

Synopsis: Checks whether a string contains a given substring.

Declaration:
```
function AnsiContainsStr(const AText: string; const ASubText: string)
                         : Boolean
```

Visibility: default

Description: `AnsiContainsString` checks whether `AText` contains `ASubText`, and returns `True` if this is the case, or returns `False` otherwise. The search is performed case-sensitive.

Errors: None

See also: AnsiContainsText (1327), AnsiEndsStr (1327), AnsiIndexStr (1327), AnsiStartsStr (1331)

74.3.4 AnsiContainsText

Synopsis: Check whether a string contains a certain substring, ignoring case.

Declaration:
```
function AnsiContainsText(const AText: string; const ASubText: string)
                                  : Boolean
```

Visibility: default

Description: `AnsiContainsString` checks whether `AText` contains `ASubText`, and returns `True` if this is the case, or returns `False` otherwise. The search is performed case-insensitive.

See also: AnsiContainsStr (1326), AnsiEndsText (1327), AnsiIndexText (1328), AnsiStartsText (1331)

74.3.5 AnsiEndsStr

Synopsis: Check whether a string ends with a certain substring.

Declaration:
```
function AnsiEndsStr(const ASubText: string; const AText: string)
                            : Boolean
```

Visibility: default

Description: `AnsiEndsStr` checks `AText` to see whether it ends with `ASubText`, and returns `True` if it does, `False` if not. The check is performed case-sensitive. Basically, it checks whether the position of `ASubText` equals the length of `AText` minus the length of `ASubText` plus one.

Errors: None.

See also: AnsiEndsText (1327), AnsiStartsStr (1331), AnsiIndexStr (1327), AnsiContainsStr (1326)

74.3.6 AnsiEndsText

Synopsis: Check whether a string ends with a certain substring, ignoring case.

Declaration:
```
function AnsiEndsText(const ASubText: string; const AText: string)
                             : Boolean
```

Visibility: default

Description: `AnsiEndsStr` checks `AText` to see whether it ends with `ASubText` , and returns `True` if it does, `False` if not. The check is performed case-insensitive. Basically, it checks whether the position of `ASubText` equals the length of `AText` minus the length of `ASubText` plus one.

Errors: None

See also: AnsiStartsText (1331), AnsiEndsStr (1327), AnsiIndexText (1328), AnsiContainsText (1327)

74.3.7 AnsiIndexStr

Synopsis: Searches, observing case, for a string in an array of strings.

Declaration:
```
function AnsiIndexStr(const AText: string;
                           const AValues: Array of string) : Integer
```

Visibility: default

Description: `AnsiIndexStr` matches `AText` against each string in `AValues`. If a match is found, the corresponding index (zero-based) in the `AValues` array is returned. If no match is found, -1 is returned. The strings are matched observing case.

Errors: None.

See also: AnsiIndexText (1328), AnsiMatchStr (1328), AnsiMatchText (1329)

74.3.8 AnsiIndexText

Synopsis: Searches, case insensitive, for a string in an array of strings.

Declaration:
```
function AnsiIndexText(const AText: string;
                       const AValues: Array of string) : Integer
```

Visibility: default

Description: `AnsiIndexText` matches `AText` against each string in `AValues`. If a match is found, the corresponding index (zero-based) in the `AValues` array is returned. If no match is found, -1 is returned. The strings are matched ignoring case.

Errors: None

See also: AnsiIndexStr (1327), AnsiMatchStr (1328), IndexStr (1341), MatchStr (1344), AnsiMatchText (1329)

74.3.9 AnsiLeftStr

Synopsis: Copies a number of characters starting at the left of a string.

Declaration:
```
function AnsiLeftStr(const AText: AnsiString; const ACount: SizeInt)
                     : AnsiString
```

Visibility: default

Description: `AnsiLeftStr` returns the `ACount` leftmost characters from `AText`. If `ACount` is larger than the length of `AText`, only as much characters as available in `AText` will be copied. If `ACount` is zero or negative, no characters will be copied. The characters are counted as characters, not as Bytes.

This function corresponds to the Visual Basic `LeftStr` function.

Errors: None.

See also: AnsiMidStr (1329), AnsiRightStr (1331), LeftStr (1344), RightStr (1351), MidStr (1345), LeftBStr (1343), RightBStr (1351), MidBStr (1345)

74.3.10 AnsiMatchStr

Synopsis: Check whether a string occurs in an array of strings, observing case.

Declaration:
```
function AnsiMatchStr(const AText: string;
                      const AValues: Array of string) : Boolean
```

Visibility: default

Description: `AnsiMatchStr` matches `AText` against each string in `AValues`. If a match is found, it returns `True`, otherwise `False` is returned. The strings are matched observing case.

This function simply calls AnsiIndexStr (1327) and checks whether it returns -1 or not.

74.3.11 AnsiMatchText

Synopsis: Check whether a string occurs in an array of strings, disregarding case.

Declaration:
```
function AnsiMatchText(const AText: string;
                       const AValues: Array of string) : Boolean
```

Visibility: default

Description: `AnsiIndexStr` matches `AText` against each string in `AValues`. If a match is found, it returns `True`, otherwise `False` is returned. The strings are matched ignoring case.

This function simply calls AnsiIndexText (1328) and checks whether it returns -1 or not.

74.3.12 AnsiMidStr

Synopsis: Returns a number of characters copied from a given location in a string.

Declaration:
```
function AnsiMidStr(const AText: AnsiString; const AStart: SizeInt;
                    const ACount: SizeInt) : AnsiString
```

Visibility: default

Description: `AnsiMidStr` returns `ACount` characters from `AText`, starting at position `AStart`. If `AStart+ACount` is larger than the length of `AText`, only as much characters as available in `AText` (starting from `AStart`) will be copied. If `ACount` is zero or negative, no characters will be copied. The characters are counted as characters, not as Bytes.

This function corresponds to the Visual Basic `MidStr` function.

Errors: None

See also: AnsiLeftStr (1328), AnsiRightStr (1331), LeftStr (1344), RightStr (1351), MidStr (1345), LeftBStr (1343), RightBStr (1351), MidBStr (1345)

74.3.13 AnsiProperCase

Synopsis: Pretty-Print a string: make lowercase and capitalize first letters of words.

Declaration:
```
function AnsiProperCase(const S: string; const WordDelims: TSysCharSet)
                        : string
```

Visibility: default

Description: `AnsiProperCase` converts `S` to an all lowercase string, but capitalizes the first letter of every word in the string, and returns the resulting string. When searching for words, the characters in `WordDelimiters` are used to determine the boundaries of words. The constant StdWordDelims (1324) can be used for this.

74.3.14 AnsiReplaceStr

Synopsis: Search and replace all occurrences of a string, case sensitive.

Declaration:
```
function AnsiReplaceStr(const AText: string; const AFromText: string;
                        const AToText: string) : string
```

Visibility: default

Description: `AnsiReplaceString` searches `AText` for all occurrences of the string `AFromText` and replaces them with `AToText`, and returns the resulting string. The search is performed observing case.

Errors: None.

See also: AnsiReplaceText (1330), SearchBuf (1353)

74.3.15 AnsiReplaceText

Synopsis: Search and replace all occurrences of a string, case insensitive.

Declaration:
```
function AnsiReplaceText(const AText: string; const AFromText: string;
                         const AToText: string) : string
```

Visibility: default

Description: `AnsiReplaceString` searches `AText` for all occurrences of the string `AFromText` and replaces them with `AToText` , and returns the resulting string. The search is performed ignoring case.

Errors: None.

See also: AnsiReplaceStr (1329), SearchBuf (1353)

74.3.16 AnsiResemblesText

Synopsis: Check whether 2 strings resemble each other.

Declaration:
```
function AnsiResemblesText(const AText: string; const AOther: string)
                           : Boolean
```

Visibility: default

Description: `AnsiResemblesText` will check whether AnsiResemblesProc (1323) is set. If it is not set, `False` is returned. If it is set, `AText` and `AOtherText` are passed to it and it's result is returned.

Errors: None.

See also: AnsiResemblesProc (1323), SoundexProc (1354)

74.3.17 AnsiReverseString

Synopsis: Reverse the letters in a string.

Declaration:
```
function AnsiReverseString(const AText: AnsiString) : AnsiString
```

Visibility: default

Description: `AnsiReverseString` returns a string with all characters of `AText` in reverse order.

if the result of this function equals `AText`, `AText` is called a palindrome.

Errors: None.

74.3.18 AnsiRightStr

Synopsis: Copies a number of characters starting at the right of a string.

Declaration:
```
function AnsiRightStr(const AText: AnsiString; const ACount: SizeInt)
                      : AnsiString
```

Visibility: default

Description: `AnsiLeftStr` returns the `ACount` rightmost characters from `AText`. If `ACount` is larger than the length of `AText`, only as much characters as available in `AText` will be copied. If `ACount` is zero or negative, no characters will be copied. The characters are counted as characters, not as Bytes.

This function corresponds to the Visual Basic `RightStr` function.

Errors: None.

See also: AnsiLeftStr (1328), AnsiMidStr (1329), LeftStr (1344), RightStr (1351), MidStr (1345), LeftBStr (1343), RightBStr (1351), MidBStr (1345)

74.3.19 AnsiStartsStr

Synopsis: Check whether a string starts with a given substring, observing case.

Declaration:
```
function AnsiStartsStr(const ASubText: string; const AText: string)
                       : Boolean
```

Visibility: default

Description: `AnsiStartsStr` checks `AText` to see whether it starts with `ASubText` , and returns `True` if it does, `False` if not. The check is performed case-sensitive. Basically, it checks whether the position of `ASubText` equals 1.

See also: AnsiEndsStr (1327), AnsiStartsStr (1331), AnsiIndexStr (1327), AnsiContainsStr (1326)

74.3.20 AnsiStartsText

Synopsis: Check whether a string starts with a given substring, ignoring case.

Declaration:
```
function AnsiStartsText(const ASubText: string; const AText: string)
                        : Boolean
```

Visibility: default

Description: `AnsiStartsText` checks `AText` to see whether it starts with `ASubText` , and returns `True` if it does, `False` if not. The check is performed case-insensitive. Basically, it checks whether the position of `ASubText` equals 1.

Errors: None.

See also: AnsiEndsText (1327), AnsiStartsStr (1331), AnsiIndexText (1328), AnsiContainsText (1327)

74.3.21 BinToHex

Synopsis: Convert a binary buffer to a hexadecimal string.

Declaration:
```
procedure BinToHex(const BinBuffer: TBytes; BinBufOffset: Integer;
                   var HexBuffer: TBytes; HexBufOffset: Integer;
                   Count: Integer);  Overload
procedure BinToHex(BinValue: Pointer; HexValue: PWideChar;
                   BinBufSize: Integer);  Overload
procedure BinToHex(const BinValue; HexValue: PWideChar;
                   BinBufSize: Integer);  Overload
procedure BinToHex(BinValue: PAnsiChar; HexValue: PAnsiChar;
                   BinBufSize: Integer);  Overload
procedure BinToHex(BinValue: PAnsiChar; HexValue: PWideChar;
                   BinBufSize: Integer);  Overload
procedure BinToHex(const BinValue; HexValue: PAnsiChar;
                   BinBufSize: Integer);  Overload
procedure BinToHex(BinValue: Pointer; HexValue: PAnsiChar;
                   BinBufSize: Integer);  Overload
```

Visibility: default

Description: `BinToHex` converts the byte values in `BinValue` to a string consisting of 2-charachter hexadecimal strings in `HexValue`. `BufSize` specifies the length of `BinValue`, which means that `HexValue` must have size `2*BufSize`.

For example a buffer containing the byte values 255 and 0 will be converted to FF00.

Errors: No length checking is done, so if an invalid size is specified, an exception may follow.

See also: HexToBin (1340)

74.3.22 ContainsStr

Synopsis: Check whether one text contains another (case sensitive).

Declaration:
```
function ContainsStr(const AText: string; const ASubText: string)
                           : Boolean
```

Visibility: default

Description: `ContainsStr` is an alias for AnsiContainsStr (1326)

See also: AnsiContainsStr (1326)

74.3.23 ContainsText

Synopsis: Check whether one text contains another (case insensitive).

Declaration:
```
function ContainsText(const AText: string; const ASubText: string)
                           : Boolean
```

Visibility: default

Description: `ContainsText` is an alias for AnsiContainsText (1327)

See also: AnsiContainsText (1327)

74.3.24 Copy2Space

Synopsis: Returns all characters in a string till the first space character (not included).

Declaration: `function Copy2Space(const S: string) : string`

Visibility: default

Description: `Copy2Space` determines the position of the first space in the string `S` and returns all characters up to this position. The space character itself is not included in the result string. The string `S` is left untouched. If there is no space in `S`, then the whole string `S` is returned.

This function simply calls Copy2Symb (1333) with the space (ASCII code 32) as the symbol argument.

Errors: None.

See also: Copy2Symb (1333), Copy2SpaceDel (1333)

74.3.25 Copy2SpaceDel

Synopsis: Deletes and returns all characters in a string till the first space character (not included).

Declaration: `function Copy2SpaceDel(var S: string) : string`

Visibility: default

Description: `Copy2SpaceDel` determines the position of the first space in the string `S` and returns all characters up to this position. The space character itself is not included in the result string. All returned characters, including the space, are deleted from the string `S`, after which it is right-trimmed. If there is no space in `S`, then the whole string `S` is returned, and `S` itself is emptied.

This function simply calls Copy2SymbDel (1334) with the space (ASCII code 32) as the symbol argument.

Errors: None.

See also: Copy2SymbDel (1334), Copy2Space (1333)

74.3.26 Copy2Symb

Synopsis: Returns all characters in a string till a given character (not included).

Declaration: `function Copy2Symb(const S: string; Symb: Char) : string`

Visibility: default

Description: `Copy2Symb` determines the position of the first occurrence of `Symb` in the string `S` and returns all characters up to this position. The `Symb` character itself is not included in the result string. The string `S` is left untouched. If `Symb` does not appear in `S`, then the whole of `S` is returned.

Errors: None.

See also: Copy2Space (1333), Copy2SymbDel (1334)

74.3.27 Copy2SymbDel

Synopsis: Deletes and returns all characters in a string till a given character (not included).

Declaration: `function Copy2SymbDel(var S: string; Symb: Char) : string`

Visibility: default

Description: `Copy2SymbDel` determines the position of the first occurrence of `Symb` in the string `S` and returns all characters up to this position. The `Symb` character itself is not included in the result string. All returned characters and the `Symb` character, are deleted from the string `S`, after which it is right-trimmed. If `Symb` does not appear in `S`, then the whole of `S` is returned, and `S` itself is emptied.

Errors: None.

See also: Copy2SpaceDel (1333), Copy2Symb (1333)

74.3.28 Dec2Numb

Synopsis: Convert a decimal number to a string representation, using given a base.

Declaration: `function Dec2Numb(N: LongInt; Len: Byte; Base: Byte) : string`

Visibility: default

Description: `Dec2Numb` converts `N` to its representation using base `Base`. `N` must be a positive integer. The resulting string is left-padded with zeroes till it has length `Len`. `Base` must be in the range 2-36 to be meaningful, but no checking on this is performed.

Errors: If `Base` is out of range, the resulting string will contain unreadable (non-alphanumeric) characters.

See also: Hex2Dec (1340), IntToBin (1342), intToRoman (1342), RomanToInt (1351)

74.3.29 DecodeSoundexInt

Synopsis: Decodes the integer representation of a soundex code and returns the original soundex code.

Declaration: `function DecodeSoundexInt(AValue: Integer) : string`

Visibility: default

Description: `DecodeSoundexInt` converts the integer value `AValue` to a soundex string. It performs the reverse operation of the SoundexInt (1354) function. The result is the soundex string corresponding to `AValue`. .

Errors: None.

See also: SoundexInt (1354), DecodeSoundexWord (1334), Soundex (1353)

74.3.30 DecodeSoundexWord

Synopsis: Decodes the word-sized representation of a soundex code and returns the original soundex code.

Declaration: `function DecodeSoundexWord(AValue: Word) : string`

Visibility: default

Description: `DecodeSoundexWord` converts the integer value `AValue` to a soundex string. It performs the reverse operation of the SoundexWord (1355) function. The result is the soundex string corresponding to `AValue` .

Errors: None.

See also: SoundexInt (1354), DecodeSoundexInt (1334), Soundex (1353)

74.3.31 DelChars

Synopsis: Delete all occurrences of a given character from a string.

Declaration:
```
function DelChars(const S: string; Chr: Char) : string
function DelChars(const S: string; Chars: TSysCharSet) : string
```

Visibility: default

Description: `DelChars` returns a copy of `S` with all `Chr` characters removed from it.

Errors: None.

See also: DelSpace (1335), DelSpace1 (1335)

74.3.32 DelSpace

Synopsis: Delete all occurrences of a space from a string.

Declaration:
```
function DelSpace(const S: string) : string
```

Visibility: default

Description: `DelSpace` returns a copy of `S` with all spaces (ASCII code 32) removed from it.

Errors: None.

See also: DelChars (1335), DelSpace1 (1335)

74.3.33 DelSpace1

Synopsis: Reduces sequences of space characters to 1 space character.

Declaration:
```
function DelSpace1(const S: string) : string
```

Visibility: default

Description: `DelSpace1` returns a copy of `S` with all sequences of spaces reduced to 1 space.

Errors: None.

See also: DelChars (1335), DelSpace (1335)

74.3.34 DupeString

Synopsis: Creates and concatenates N copies of a string.

Declaration:
```
function DupeString(const AText: string; ACount: Integer) : string
```

Visibility: default

Description: `DupeString` returns a string consisting of `ACount` concatenations of `AText`. Thus

```
DupeString('1234567890',3);
```

will produce a string

```
'12345678901234567890123456789O'
```

If `aCount<=0` then the result is an empty string.

Errors: If you specify a too large `aCount` (so the resulting string would require more memory than allowed on your system) then an `EOutOfMemory` exception can be raised.

74.3.35 EndsStr

Synopsis: Check whether one string ends with another.

Declaration:
```
function EndsStr(const ASubText: string; const AText: string) : Boolean
```

Visibility: default

Description: `StarsText` checks whether `aText` ends with `aSubText` and returns `True` if it does. i.e. it returns true if the last characters of `aText` are `aSubText`. It follows that the length of `aText` must be at least the length of `aSubText`. The comparison is made case-sensitive. If you wish to compare case-insensitively, use EndsText (1336) instead.

See also: AnsiEndsStr (1327), EndsText (1336), StartsText (1356)

74.3.36 EndsText

Synopsis: Check whether one text ends with another.

Declaration:
```
function EndsText(const ASubText: string; const AText: string) : Boolean
```

Visibility: default

Description: `StarsText` checks whether `aText` ends with `aSubText` and returns `True` if it does. i.e. it returns true if the last characters of `aText` are `aSubText`. It follows that the length of `aText` must be at least the length of `aSubText`. The comparison is made case-insensitive. If you wish to compare case-sensitively, use EndsStr (1336) instead.

See also: AnsiEndsStr (1327), EndsStr (1336), StartsText (1356)

74.3.37 ExtractDelimited

Synopsis: Extract the N-th delimited part from a string.

Declaration:
```
function ExtractDelimited(N: Integer; const S: string;
                          const Delims: TSysCharSet) : string
```

Visibility: default

Description: `ExtractDelimited` extracts the `N`-th part from the string `S`. The set of characters in `Delims` are used to mark part boundaries. When a delimiter is encountered, a new part is started and the old part is ended. Another way of stating this is that any (possibly empty) series of characters not in `Delims`, situated between 2 characters in `Delims`, it is considered as piece of a part. This means that if 2 delimiter characters appear next to each other, there is an empty part between it. If an `N`-th part cannot be found, an empty string is returned. However, unlike ExtractWord (1337), an empty string is a valid return value, i.e. a part can be empty.

The predefined constant StdWordDelims (1324) can be used for the `Delims` argument. The predefined constant Brackets (1323) would be better suited the `Delims` argument e.g. in case factors in a mathematical expression are searched.

Errors: None.

See also: ExtractSubStr (1337), ExtractWord (1337), ExtractWordPos (1337)

74.3.38 ExtractSubstr

Synopsis: Extract a word from a string, starting at a given position in the string.

Declaration:
```
function ExtractSubstr(const S: string; var Pos: Integer;
                       const Delims: TSysCharSet) : string
```

Visibility: default

Description: `ExtractSubStr` returns all characters from `S` starting at position `Pos` till the first character in `Delims`, or till the end of `S` is reached. The delimiter character is not included in the result. `Pos` is then updated to point to the next first non-delimiter character in `S`. If `Pos` is larger than the Length of `S`, an empty string is returned.

The predefined constant StdWordDelims (1324) can be used for the `Delims` argument.

Errors: None.

See also: ExtractDelimited (1336), ExtractWord (1337), ExtractWordPos (1337)

74.3.39 ExtractWord

Synopsis: Extract the N-th word out of a string.

Declaration:
```
function ExtractWord(N: Integer; const S: string;
                     const WordDelims: TSysCharSet) : string
```

Visibility: default

Description: `ExtractWord` extracts the `N`-th word from the string `S`. The set of characters in `WordDelims` are used to mark word boundaries. A word is defined as any non-empty sequence of characters which are not present in `WordDelims`: if a character is not in `WordDelims`, it is considered as part of a word. If an `N`-th word cannot be found, an empty string is returned.

Unlike ExtractDelimited (1336), an empty string is not a valid return value, i.e. is not a word. If an empty string is returned, the index `N` was out of range.

The predefined constant StdWordDelims (1324) can be used for the `WordDelims` argument.

Errors: None.

See also: ExtractWordPos (1337), ExtractSubStr (1337), ExtractDelimited (1336), IsWordPresent (1343), WordCount (1359), WordPosition (1360)

74.3.40 ExtractWordPos

Synopsis: Extract a word from a string, and return the position where it was located in the string.

Declaration:
```
function ExtractWordPos(N: Integer; const S: string;
                        const WordDelims: TSysCharSet; out Pos: Integer)
                         : string
```

Visibility: default

Description: `ExtractWordPos` extracts the `N`-th word from the string `S` and returns the position of this word in `Pos`. The set of characters in `WordDelims` are used to mark word boundaries. A word is defined as any non-empty sequence of characters which are not present in `WordDelims` : if a character is not in `WordDelims` , it is considered as part of a word. If an `N` -th word cannot be found, an empty string is returned and `Pos` is zero.

Unlike ExtractDelimited (1336), an empty string is not a valid return value, i.e. is not a word. If an empty string is returned, the index `N` was out of range.

The predefined constant StdWordDelims (1324) can be used for the `WordDelims` argument.

Errors: None.

See also: ExtractWord (1337), ExtractSubStr (1337), IsWordPresent (1343), WordCount (1359), WordPosition (1360)

74.3.41 FindMatchesBoyerMooreCaseInSensitive

Synopsis: Find case-insensitive matches of a string using a Boyer-Moore algorithm.

Declaration:
```
function FindMatchesBoyerMooreCaseInSensitive(const S: PChar;
                                              const OldPattern: PChar;
                                              const SSize: SizeInt;
                                              const OldPatternSize: SizeInt;
                                              out aMatches: SizeIntArray;
                                              const aMatchAll: Boolean)
                                               : Boolean
function FindMatchesBoyerMooreCaseInSensitive(const S: string;
                                              const OldPattern: string;
                                              out aMatches: SizeIntArray;
                                              const aMatchAll: Boolean)
                                               : Boolean
```

Visibility: default

Description: `FindMatchesBoyerMooreCaseInSensitive` finds occurrences of `OldPattern` (with length `OldPatternSize`) in `S` (with length `SSize`). The search is performed case-insensitively, and all (zero based) positions are reported in `aMatches`. If `aMatchAll` is `True`, all positions will be reported. If `aMatchAll` is `False`, only the first position is reported.

Errors: None.

See also: FindMatchesBoyerMooreCaseSensitive (1338), StringReplace (1357)

74.3.42 FindMatchesBoyerMooreCaseSensitive

Synopsis: Find case-sensitive matches of a string using a Boyer-Moore algorithm.

Declaration:
```
function FindMatchesBoyerMooreCaseSensitive(const S: PChar;
                                            const OldPattern: PChar;
                                            const SSize: SizeInt;
                                            const OldPatternSize: SizeInt;
                                            out aMatches: SizeIntArray;
                                            const aMatchAll: Boolean)
                                             : Boolean
function FindMatchesBoyerMooreCaseSensitive(const S: string;
                                            const OldPattern: string;
```

```
                                                    out aMatches: SizeIntArray;
                                                    const aMatchAll: Boolean)
                                                     : Boolean
```

Visibility: default

Description: `FindMatchesBoyerMooreCaseSensitive` finds occurrences of `OldPattern` (with length `OldPatternSize`) in `S` (with length `SSize`). The search is performed case-sensitively, and all (zero based) positions are reported in `aMatches`. If `aMatchAll` is `True`, all positions will be reported. If `aMatchAll` is `False`, only the first position is reported.

Errors: None.

See also: FindMatchesBoyerMooreCaseInSensitive (1338), StringReplace (1357)

74.3.43 FindPart

Synopsis: Search for a substring in a string, using wildcards.

Declaration:
```
function FindPart(const HelpWilds: string; const InputStr: string)
                        : SizeInt
```

Visibility: default

Description: `FindPart` searches the string `InputStr` and returns the first string that matches the wildcards specification in `HelpWilds` . If no match is found, and empty string is returned. Currently, the only valid wildcards is the "?" character.

Errors: None.

See also: SearchBuf (1353)

74.3.44 GetCmdLineArg

Synopsis: Returns the command-line argument following the given switch.

Declaration:
```
function GetCmdLineArg(const Switch: string; SwitchChars: TSysCharSet)
                              : string
```

Visibility: default

Description: `GetCmdLineArg` returns the value for the `Switch` option on the command-line, if any is given. Command-line arguments are considered switches if they start with one of the characters in the `SwitchChars` set. The value is the command-line argument following the switch command-line argument.

Gnu-style (long) Options of the form switch=value are not supported.

The StdSwitchChars (1324) constant can be used as value for the `SwitchChars` parameter.

Errors: The GetCmdLineArg does not check whether the value of the option does not start with a switch character. i.e.

```
myprogram -option1 -option2
```

will result in "-option2" as the result of the `GetCmdLineArg` call for option1.

See also: StdSwitchChars (1324)

74.3.45 Hex2Dec

Synopsis: Converts a hexadecimal string to a decimal value.

Declaration:
```
function Hex2Dec(const S: string) : LongInt
```

Visibility: default

Description: `Hex2Dec` converts the hexadecimal value in the string `S` to its decimal value. Unlike the standard `Val`or `StrToInt` functions, there need not be a $ sign in front of the hexadecimal value to indicate that it is indeed a hexadecimal value.

Errors: If `S` does not contain a valid hexadecimal value, an `EConvertError` exception will be raised.

See also: Dec2Numb (1334), IntToBin (1342), intToRoman (1342), RomanToInt (1351)

74.3.46 Hex2Dec64

Synopsis: Convert hexadecimal value to 64-bit integer.

Declaration:
```
function Hex2Dec64(const S: string) : Int64
```

Visibility: default

Description: `Hex2Dec64` converts the hexadecimal number in S to a 64-bit value and returns the result. It is equivalent to `StrToInt('$'+S)`.

See also: Hex2Dec (1340)

74.3.47 HexToBin

Synopsis: Convert a hexadecimal string to a binary buffer.

Declaration:
```
function HexToBin(HexValue: PChar; BinValue: PChar; BinBufSize: Integer)
                        : Integer
```

Visibility: default

Description: `HexToBin` scans the hexadecimal string representation in `HexValue` and transforms every 2 character hexadecimal number to a byte and stores it in `BinValue`. The buffer size is the size of the binary buffer. Scanning will stop if the size of the binary buffer is reached or when an invalid character is encountered. The return value is the number of stored bytes.

Errors: No length checking is done, so if an invalid size is specified, an exception may follow.

See also: BinToHex (1331)

74.3.48 IfThen

Synopsis: Returns one of two strings, depending on a boolean expression.

Declaration:
```
function IfThen(AValue: Boolean; const ATrue: string;
                      const AFalse: string) : string;  Overload
function IfThen(AValue: Boolean; const ATrue: TStringDynArray;
                      const AFalse: TStringDynArray) : TStringDynArray
                      ;  Overload
```

Visibility: default

Description: `IfThen` returns `ATrue` if `AValue` is `True`, and returns `AFalse` if `AValue` is false.

Errors: None.

See also: AnsiMatchStr (1328), AnsiMatchText (1329)

74.3.49 in(string,):Boolean

Synopsis: Check presence of string in an array of strings.

Declaration:
```
operator in(const AText: string; const AValues: Array of string)
                  : Boolean
```

Visibility: default

Description: This is the (string) operator equivalent of IndexStr (1341)

See also: IndexStr (1341)

74.3.50 in(UnicodeString,):Boolean

Synopsis: Check presence of string in an array of strings.

Declaration:
```
operator in(const AText: UnicodeString;
                const AValues: Array of UnicodeString) : Boolean
```

Visibility: default

Description: This is the (unicode string) operator equivalent of IndexStr (1341)

See also: IndexStr (1341)

74.3.51 IndexStr

Synopsis: Searches, observing case, for a string in an array of strings.

Declaration:
```
function IndexStr(const AText: UnicodeString;
                  const AValues: Array of UnicodeString) : Integer
function IndexStr(const AText: string; const AValues: Array of string)
                  : Integer
```

Visibility: default

Description: `IndexStr` matches `AText` against each Unicode string in `AValues`. If a match is found, the corresponding index (zero-based) in the `AValues` array is returned. If no match is found, -1 is returned. The strings are matched observing case.

Errors: None.

See also: AnsiIndexText (1328), MatchStr (1344), AnsiIndexStr (1327)

74.3.52 IndexText

Synopsis: Index of text in a list of values.

Declaration:
```
function IndexText(const AText: string; const AValues: Array of string)
                    : Integer
function IndexText(const AText: UnicodeString;
                   const AValues: Array of UnicodeString) : Integer
```

Visibility: default

Description: `IndexText` returns the index of the string in the array `aValues` that matches `aText`, and -1 otherwise. The comparison is done case insensitively. If you wish to compare case sensitively, use IndexStr (1341) instead.

See also: IndexStr (1341), MatchText (1344), AnsiIndexText (1328)

74.3.53 IntToBin

Synopsis: Converts an integer to a binary string representation, inserting spaces at fixed locations.

Declaration:
```
function IntToBin(Value: LongInt; Digits: Integer; Spaces: Integer)
                  : string
function IntToBin(Value: LongInt; Digits: Integer) : string
function IntToBin(Value: Int64; Digits: Integer) : string
```

Visibility: default

Description: `IntToBin` converts `Value` to a string with it's binary (base 2) representation. The resulting string contains `Digits` digits, with spaces inserted every `Spaces` digits. `Spaces` equal to zero yields a result without spacing. If `Digits` is larger than 32, it is truncated to 32.

See also: Hex2Dec (1340), IntToRoman (1342)

74.3.54 IntToRoman

Synopsis: Represent an integer with roman numerals.

Declaration:
```
function IntToRoman(Value: LongInt) : string
```

Visibility: default

Description: `IntToRoman` converts `Value` to a string with the Roman representation of `Value`. Number up to 1 million can be represented this way.

Errors: None.

See also: RomanToInt (1351), Hex2Dec (1340), IntToBin (1342)

74.3.55 IsEmptyStr

Synopsis: Check whether a string is empty, disregarding whitespace characters.

Declaration:
```
function IsEmptyStr(const S: string; const EmptyChars: TSysCharSet)
                    : Boolean
```

Visibility: default

Description: `IsEmptyStr` returns `True` if the string `S` only contains characters whitespace characters, all characters in `EmptyChars` are considered whitespace characters. If a character not present in `EmptyChars` is found in `S`, `False` is returned.

Errors: None.

See also: IsWild (1343), FindPart (1339), IsWordPresent (1343)

74.3.56 IsWild

Synopsis: Check whether a string matches a wildcard search expression.

Declaration:
```
function IsWild(InputStr: string; Wilds: string; IgnoreCase: Boolean)
                        : Boolean
```

Visibility: default

Description: `IsWild` checks `InputStr` for the presence of the `Wilds` string. `Wilds` may contain "?" and "*" wildcard characters, which have their usual meaning: "*" matches any series of characters, possibly empty. "?" matches any single character. The function returns `True` if a string is found that matches `Wilds`, `False` otherwise.

If `IgnoreCase` is `True`, the non-wildcard characters are matched case insensitively. If it is `False`, case is observed when searching.

Errors: None.

See also: SearchBuf (1353), FindPart (1339)

74.3.57 IsWordPresent

Synopsis: Check for the presence of a word in a string.

Declaration:
```
function IsWordPresent(const W: string; const S: string;
                             const WordDelims: TSysCharSet) : Boolean
```

Visibility: default

Description: `IsWordPresent` checks for the presence of the word `W` in the string `S`. Words are delimited by the characters found in `WordDelims`. The function returns `True` if a match is found, `False` otherwise. The search is performed case sensitive.

This function is equivalent to the SearchBuf (1353) function with the `soWholeWords` option specified.

Errors: None.

See also: SearchBuf (1353)

74.3.58 LeftBStr

Synopsis: Copies Count characters starting at the left of a string.

Declaration:
```
function LeftBStr(const AText: AnsiString; const AByteCount: SizeInt)
                        : AnsiString
```

Visibility: default

Description: `LeftBStr` returns a string containing the leftmost `AByteCount` bytes from the string `AText` . If `AByteCount` is larger than the length (in bytes) of `AText` , only as many bytes as available are returned.

Errors: None.

See also: LeftStr (1344), AnsiLeftStr (1328), RightBStr (1351), MidBStr (1345)

74.3.59 LeftStr

Synopsis: Copies Count characters starting at the left of a string.

Declaration:
```
function LeftStr(const AText: AnsiString; const ACount: SizeInt)
                  : AnsiString
function LeftStr(const AText: WideString; const ACount: SizeInt)
                  : WideString
```

Visibility: default

Description: `LeftStr` returns a string containing the leftmost `ACount` characters from the string `AText` . If `ACount` is larger than the length (in characters) of `AText` , only as many characters as available are returned.

Errors: None.

See also: LeftBStr (1343), AnsiLeftStr (1328), RightStr (1351), MidStr (1345)

74.3.60 MatchStr

Synopsis: Check whether a string occurs in an array of strings, observing case.

Declaration:
```
function MatchStr(const AText: UnicodeString;
                  const AValues: Array of UnicodeString) : Boolean
function MatchStr(const AText: string; const AValues: Array of string)
                   : Boolean
```

Visibility: default

Description: `MatchStr` matches `AText` against each Unicode string in `AValues`. If a match is found, it returns `True`, otherwise `False` is returned. The strings are matched observing case.

This function simply calls IndexStr (1341) and checks whether it returns -1 or not.

74.3.61 MatchText

Synopsis: Check if a string is in a list of values.

Declaration:
```
function MatchText(const AText: string; const AValues: Array of string)
                    : Boolean
function MatchText(const AText: UnicodeString;
                   const AValues: Array of UnicodeString) : Boolean
```

Visibility: default

Description: `MatchText` returns `True` if `aText` equals one of the strings in `aValues`. The comparison is done case insensitively. If you wish to compare case sensitively, use MatchStr (1344) instead.

See also: MatchStr (1344), AnsiMatchText (1329), IndexText (1342)

74.3.62 MidBStr

Synopsis: Copies a number of characters starting at a given position in a string.

Declaration:
```
function MidBStr(const AText: AnsiString; const AByteStart: SizeInt;
                 const AByteCount: SizeInt) : AnsiString
```

Visibility: default

Description: `MidBStr` returns a string containing the first `AByteCount` bytes from the string `AText` starting at position `AByteStart`. If `AByteStart+AByteCount` is larger than the length (in bytes) of `AText`, only as many bytes as available are returned. If `AByteStart` is less than 1 or larger than the length of `AText`, then no characters are returned.

Errors: None.

See also: LeftBStr (1343), AnsiMidStr (1329), RightBStr (1351), MidStr (1345)

74.3.63 MidStr

Synopsis: Copies a number of characters starting at a given position in a string.

Declaration:
```
function MidStr(const AText: AnsiString; const AStart: SizeInt;
                const ACount: SizeInt) : AnsiString
function MidStr(const AText: WideString; const AStart: SizeInt;
                const ACount: SizeInt) : WideString
```

Visibility: default

Description: `MidStr` returns a string containing the first `ACount` bytes from the string `AText` starting at position `AStart`. If `AStart+ACount` is larger than the length (in characters) of `AText`, only as many characters as available are returned. If `AStart` is less than 1 or larger than the length of `AText`, then no characters are returned.

This function is equivalent to the standard `Copy` function, and is provided for completeness only.

Errors: None.

See also: LeftStr (1344), AnsiMidStr (1329), RightStr (1351), MidBStr (1345)

74.3.64 NaturalCompareText

Synopsis: Compare using natural sort.

Declaration:
```
function NaturalCompareText(const S1: string; const S2: string)
                            : Integer
function NaturalCompareText(const Str1: string; const Str2: string;
                            const ADecSeparator: Char;
                            const AThousandSeparator: Char) : Integer
```

Visibility: default

Description: `NaturalCompareText` will compare 2 strings and return one of the following values:

-1 If `S1` comes before `S2`

0 If `S1` equals `S2`

1 If `S1` comes after `S2`

When S1 and S2 are integer or floating point values, the actual values are compared. Thus '12' will come after '2'. If either of the values is not a valid integer or floating point values, the strings are compared case insensitively as texts using CompareText (1690)

A decimal separator and thousands separator can optionally be specified in ADecSeparator, AThousandSeparator. If none are specified, the defaults from the system unit will be used.

See also: CompareText (1323)

74.3.65 NPos

Synopsis: Returns the position of the N-th occurrence of a substring in a string.

Declaration:
```
function NPos(const C: string; S: string; N: Integer) : SizeInt
```

Visibility: default

Description: NPos checks S for the position of the N-th occurrence of C. If C occurs less than N times in S, or does not occur in S at all, 0 is returned. If N is less than 1, zero is returned.

Errors: None.

See also: WordPosition (1360), FindPart (1339), #rtl.system.Pos (1543), PosEx (1347)

74.3.66 Numb2Dec

Synopsis: Converts a string representation of a number to its numerical value, given a certain base.

Declaration:
```
function Numb2Dec(S: string; Base: Byte) : LongInt
```

Visibility: default

Description: Numb2Dec converts the number in string S to a decimal value. It assumes the number is represented using Base as the base. No checking is performed to see whether S contains a valid number using base Base.

Errors: None.

See also: Hex2Dec (1340), Numb2USA (1346)

74.3.67 Numb2USA

Synopsis: Insert thousand separators.

Declaration:
```
function Numb2USA(const S: string) : string
```

Visibility: default

Description: Numb2USA inserts thousand separators in the string S at the places where they are supposed to be, i.e. every 3 digits. The string S should contain a valid integer number, i.e. no digital number. No checking on this is done.

Errors: None.

74.3.68 PadCenter

Synopsis: Pad the string to a certain length, so the string is centered.

Declaration:
```
function PadCenter(const S: string; Len: SizeInt) : string
```

Visibility: default

Description: `PadCenter` add spaces to the left and right of the string `S` till the result reaches length `Len`. If the number of spaces to add is odd, then the extra space will be added at the end. If the string `S` has length equal to or larger than `Len`, no spaces are added, and the string `S` is returned as-is.

Errors: None.

See also: PadLeft (1347), PadRight (1347), AddChar (1326), AddCharR (1326)

74.3.69 PadLeft

Synopsis: Add spaces to the left of a string till a certain length is reached.

Declaration:
```
function PadLeft(const S: string; N: Integer) : string
```

Visibility: default

Description: `PadLeft` add spaces to the left of the string `S` till the result reaches length `Len`. If the string `S` has length equal to or larger than `Len`, no spaces are added, and the string `S` is returned as-is. The resulting string is `S`, right-justified on length `Len`.

Errors: None.

See also: PadLeft (1347), PadCenter (1347), AddChar (1326), AddCharR (1326)

74.3.70 PadRight

Synopsis: Add spaces to the right of a string till a certain length is reached.

Declaration:
```
function PadRight(const S: string; N: Integer) : string
```

Visibility: default

Description: `PadRight` add spaces to the right of the string `S` till the result reaches length `Len`. If the string `S` has length equal to or larger than `Len`, no spaces are added, and the string `S` is returned as-is. The resulting string is `S`, left-justified on length `Len`.

Errors: None.

See also: PadLeft (1347), PadCenter (1347), AddChar (1326), AddCharR (1326)

74.3.71 PosEx

Synopsis: Search for the occurrence of a character in a string, starting at a certain position.

Declaration:
```
function PosEx(const SubStr: string; const S: string; Offset: SizeUInt)
                  : SizeInt
function PosEx(const SubStr: string; const S: string) : SizeInt
function PosEx(c: Char; const S: string; Offset: SizeUInt) : SizeInt
function PosEx(const SubStr: UnicodeString; const S: UnicodeString;
              Offset: SizeUInt) : SizeInt
```

```
function PosEx(c: WideChar; const S: UnicodeString; Offset: SizeUInt)
                : SizeInt
function PosEx(const SubStr: UnicodeString; const S: UnicodeString)
                : SizeInt
```

Visibility: default

Description: `PosEx` returns the position of the first occurrence of the character `C` or the substring `SubStr` in the string `S`, starting the search at position `Offset` (default 1). If `C` or `SubStr` does not occur in `S` after the given `Offset`, zero is returned. The position Offset is also searched.

Errors: None.

See also: NPos (1346), AnsiContainsText (1327), AnsiContainsStr (1326)

74.3.72 PosSet

Synopsis: Return the position in a string of any character out of a set of characters.

Declaration:
```
function PosSet(const c: TSysCharSet; const s: ansistring) : SizeInt
function PosSet(const c: string; const s: ansistring) : SizeInt
```

Visibility: default

Description: `PosSet` returns the position in `s` of the first found character which is in the set `c`. If none of the characters in `c` is found in `s`, then `0` is returned.

Errors: None.

See also: PosEx (1347), PosSetEx (1348), #rtl.system.pos (1543), RPosEx (1352)

74.3.73 PosSetEx

Synopsis: Return the position in a string of any character out of a set of characters, starting at a certain position.

Declaration:
```
function PosSetEx(const c: TSysCharSet; const s: ansistring;
                  count: Integer) : SizeInt
function PosSetEx(const c: string; const s: ansistring; count: Integer)
                  : SizeInt
```

Visibility: default

Description: `PosSetEx` returns the position in `s` of the first found character which is in the set `c`, and starts searching at character position `Count`. If none of the characters in `c` is found in `s`, then `0` is returned.

Errors: None.

See also: PosEx (1347), PosSet (1348), #rtl.system.pos (1543), RPosEx (1352)

74.3.74 RandomFrom

Synopsis: Choose a random string from an array of strings.

Declaration:
```
function RandomFrom(const AValues: Array of string) : string;  Overload
```

Visibility: default

Description: `RandomFrom` picks at random a valid index in the array `AValues` and returns the string at that position in the array.

Errors: None.

See also: AnsiMatchStr (1328), AnsiMatchText (1329)

74.3.75 RemoveLeadingchars

Synopsis: Remove any leading characters in a set from a string.

Declaration:
```
procedure RemoveLeadingchars(var S: AnsiString; const CSet: TSysCharSet)
```

Visibility: default

Description: `Removeleadingchars` removes any starting characters from `S` that appear in the set `CSet`. It stops removing characters as soon as a character not in `CSet` is encountered. This is similar in behaviour to TrimLeft (1803) which used whitespace as the set.

Errors: None.

See also: TrimLeft (1803), RemoveTrailingChars (1349), RemovePadChars (1349), TrimLeftSet (1358)

74.3.76 RemovePadChars

Synopsis: Remove any trailing or leading characters in a set from a string.

Declaration:
```
procedure RemovePadChars(var S: AnsiString; const CSet: TSysCharSet)
```

Visibility: default

Description: `RemovePadChars` removes any leading trailing characters from `S` that appear in the set `CSet`, i.e. it starts with the last character and works its way to the start of the string, and it stops removing characters as soon as a character not in `CSet` is encountered. Then the same procedure is repeated starting from the beginning of the string. This is similar in behaviour to Trim (1803) which used whitespace as the set.

Errors: None.

See also: Trim (1803), RemoveLeadingChars (1349), RemoveTrailingChars (1349), TrimSet (1359), TrimLeftSet (1358), TrimRightSet (1358)

74.3.77 RemoveTrailingChars

Synopsis: Remove any trailing characters in a set from a string.

Declaration:
```
procedure RemoveTrailingChars(var S: AnsiString;
                              const CSet: TSysCharSet)
```

Visibility: default

Description: `RemoveTrailingChars` removes any trailing characters from `S` that appear in the set `CSet`, i.e. it starts with the last character and works its way to the start of the string. It stops removing characters as soon as a character not in `CSet` is encountered. This is similar in behaviour to TrimRight (1804) which used whitespace as the set.

See also: TrimRight (1803), RemoveLeadingChars (1349), TrimRightSet (1358)

74.3.78 ReplaceStr

Synopsis: Replace strings case-sensitively.

Declaration:
```
function ReplaceStr(const AText: string; const AFromText: string;
                    const AToText: string) : string
```

Visibility: default

Description: `ReplaceStr` is a utility function that scans `AText` and replaces all occurrences of `AFromText` with `AToText` and returns the resulting string. It simply calls StringReplace (1778) with the appropriate options.

See also: StringReplace (1778), ReplaceText (1350)

74.3.79 ReplaceText

Synopsis: Replace strings case-insensitively.

Declaration:
```
function ReplaceText(const AText: string; const AFromText: string;
                     const AToText: string) : string
```

Visibility: default

Description: `ReplaceText` is a utility function that scans `AText` and replaces all occurrences of `AFromText` (case insensitive) with `AToText` and returns the resulting string. It simply calls StringReplace (1778) with the appropriate options.

See also: StringReplace (1778), ReplaceText (1350)

74.3.80 ResemblesText

Synopsis: Check whether 2 strings resemble each other.

Declaration:
```
function ResemblesText(const AText: string; const AOther: string)
                       : Boolean
```

Visibility: default

Description: `ResemblesText` is an alias for AnsiResemblesText (1330)

See also: AnsiResemblesText (1330)

74.3.81 ReverseString

Synopsis: Reverse characters in a string.

Declaration:
```
function ReverseString(const AText: string) : string
```

Visibility: default

Description: `ReverseString` returns a string, made up of the characters in string `AText`, in reverse order.

Errors: None.

See also: RandomFrom (1348)

74.3.82 RightBStr

Synopsis: Copy a given number of characters (bytes), counting from the right of a string.

Declaration:
```
function RightBStr(const AText: AnsiString; const AByteCount: SizeInt)
                         : AnsiString
```

Visibility: default

Description: `RightBStr` returns a string containing the rightmost `AByteCount` bytes from the string `AText`. If `AByteCount` is larger than the length (in bytes) of `AText`, only as many bytes as available are returned.

Errors: None.

See also: LeftBStr (1343), AnsiRightStr (1331), RightStr (1351), MidBStr (1345)

74.3.83 RightStr

Synopsis: Copy a given number of characters, counting from the right of a string.

Declaration:
```
function RightStr(const AText: AnsiString; const ACount: SizeInt)
                       : AnsiString
function RightStr(const AText: WideString; const ACount: SizeInt)
                       : WideString
```

Visibility: default

Description: `RightStr` returns a string containing the rightmost `ACount` characters from the string `AText` . If `ACount` is larger than the length (in characters) of `AText` , only as many characters as available are returned.

Errors: None.

See also: LeftStr (1344), AnsiRightStr (1331), RightBStr (1351), MidStr (1345)

74.3.84 RomanToInt

Synopsis: Convert a string with a Roman number to it's decimal value.

Declaration:
```
function RomanToInt(const S: string;
                          Strictness: TRomanConversionStrictness) : LongInt
```

Visibility: default

Description: `RomanToInt` returns the decimal equivalent of the Roman numerals in the string `S`. Invalid characters are dropped from `S`. A negative numeral is supported as well. The level of error checking is determined by the `strictness` parameter, the values are described in the type TRomanConversionStrictness (1324).

Errors: On error, a EConvertError (1323) exception is raised.

See also: TRomanConversionStrictness (1324), IntToRoman (1342), Hex2Dec (1340), Numb2Dec (1346)

74.3.85 RomanToIntDef

Synopsis: Convert a roman numeral to an integer value.

Declaration:
```
function RomanToIntDef(const S: string; const ADefault: LongInt;
                       Strictness: TRomanConversionStrictness) : LongInt
```

Visibility: default

Description: `RomanToInt` converts the roman numeral in `S` to an integer and returns the integer value. The strictness of the conversion algorithm is determined by `Strictness`. If the conversion fails, `ADefault` is returned.

See also: TRomanConversionStrictness (1324), TryRomanToInt (1359), RomanToInt (1351), IntToRoman (1342)

74.3.86 RPos

Synopsis: Find last occurrence of substring or character in a string.

Declaration:
```
function RPos(c: Char; const S: AnsiString) : SizeInt;  Overload
function RPos(c: UnicodeChar; const S: UnicodeString) : SizeInt
            ;  Overload
function RPos(const Substr: AnsiString; const Source: AnsiString)
             : SizeInt;  Overload
function RPos(const Substr: UnicodeString; const Source: UnicodeString)
             : SizeInt;  Overload
```

Visibility: default

Description: `RPos` looks in `S` for the character `C` or the string `SubStr`. It starts looking at the end of the string, and searches towards the beginning of the string. If a match is found, it returns the position of the match.

See also: RPosEx (1352)

74.3.87 RPosEx

Synopsis: Find last occurrence substring or character in a string, starting at a certain position.

Declaration:
```
function RPosEx(C: Char; const S: AnsiString; offs: Cardinal) : SizeInt
              ;  Overload
function RPosEx(C: UnicodeChar; const S: UnicodeString; offs: Cardinal)
               : SizeInt;  Overload
function RPosEx(const Substr: AnsiString; const Source: AnsiString;
              offs: Cardinal) : SizeInt;  Overload
function RPosEx(const Substr: UnicodeString;
              const Source: UnicodeString; offs: Cardinal) : SizeInt
              ;  Overload
```

Visibility: default

Description: `RPos` looks in `S` for the character `C` or the string `SubStr`. It starts looking at position `Offs` (counted from the start of the string), and searches towards the beginning of the string. If a match is found, it returns the position of the match.

See also: RPos (1352)

74.3.88 SearchBuf

Synopsis: Search a buffer for a certain string.

Declaration:
```
function SearchBuf(Buf: PChar; BufLen: SizeInt; SelStart: SizeInt;
                   SelLength: SizeInt; SearchString: string;
                   Options: TStringSearchOptions) : PChar
function SearchBuf(Buf: PChar; BufLen: SizeInt; SelStart: SizeInt;
                   SelLength: SizeInt; SearchString: string) : PChar
```

Visibility: default

Description: `SearchBuf` searches the buffer `Buf` for the occurrence of `SearchString`. At must `Buflen` characters are searched, and the search is started at `SelStart+SelLength`. If a match is found, a pointer to the position of the match is returned. The parameter Options (1325) specifies how the search is conducted. It is a set of the following options:

Table 74.5:

Option	Effect
soDown	Searches forward, starting at the end of the selection. Default is searching up
soMatchCase	Observe case when searching. Default is to ignore case.
soWholeWord	Match only whole words. Default also returns parts of words

The standard constant WordDelimiters (1324) is used to mark the boundaries of words.

The `SelStart` parameter is zero based.

Errors: Buflen must be the real length of the string, no checking on this is performed.

See also: FindPart (1339), ExtractWord (1337), ExtractWordPos (1337), ExtractSubStr (1337), IsWordPresent (1343)

74.3.89 Soundex

Synopsis: Compute the soundex of a string.

Declaration:
```
function Soundex(const AText: string; ALength: TSoundexLength) : string
function Soundex(const AText: string) : string
```

Visibility: default

Description: `Soundex` computes a soundex code for `AText`. The resulting code will at most have `ALength` characters. The soundex code is computed according to the US system of soundex computing, which may result in inaccurate results in other languages.

Note that `AText` may not contain null characters.

Errors: None.

See also: SoundexCompare (1354), SoundexInt (1354), SoundexProc (1354), SoundexWord (1355), SoundexSimilar (1355)

74.3.90 SoundexCompare

Synopsis: Compare soundex values of 2 strings.

Declaration:
```
function SoundexCompare(const AText: string; const AOther: string;
                        ALength: TSoundexLength) : Integer
function SoundexCompare(const AText: string; const AOther: string)
                        : Integer
```

Visibility: default

Description: `SoundexCompare` computes the soundex codes of `AText` and `AOther` and feeds these to `CompareText`. It will return -1 if the soundex code of `AText` is less than the soundex code of `AOther`, 0 if they are equal, and 1 if the code of `AOther` is larger than the code of `AText`.

Errors: None.

See also: Soundex (1353), SoundexInt (1354), SoundexProc (1354), SoundexWord (1355), SoundexSimilar (1355)

74.3.91 SoundexInt

Synopsis: Soundex value as an integer.

Declaration:
```
function SoundexInt(const AText: string; ALength: TSoundexIntLength)
                    : Integer
function SoundexInt(const AText: string) : Integer
```

Visibility: default

Description: `SoundexInt` computes the Soundex (1353) code (with length `ALength`, default 4) of `AText`, and converts the code to an integer value.

Errors: None.

See also: Soundex (1353), SoundexCompare (1354), SoundexProc (1354), SoundexWord (1355), SoundexSimilar (1355)

74.3.92 SoundexProc

Synopsis: Default AnsiResemblesText implementation.

Declaration:
```
function SoundexProc(const AText: string; const AOther: string)
                     : Boolean
```

Visibility: default

Description: `SoundexProc` is the standard implementation for the AnsiResemblesText (1330) procedure: By default, `AnsiResemblesProc` is set to this function. It compares the soundex codes of `AOther` and `AText` and returns `True` if they are equal, or `False` if they are not.

Errors: None.

See also: Soundex (1353), SoundexCompare (1354), SoundexInt (1354), SoundexWord (1355), SoundexSimilar (1355)

74.3.93 SoundexSimilar

Synopsis: Check whether 2 strings have equal soundex values.

Declaration:
```
function SoundexSimilar(const AText: string; const AOther: string;
                        ALength: TSoundexLength) : Boolean
function SoundexSimilar(const AText: string; const AOther: string)
                        : Boolean
```

Visibility: default

Description: `SoundexSimilar` returns `True` if the soundex codes (with length `ALength`) of `AText` and `AOther` are equal, and `False` if they are not.

Errors: None.

See also: Soundex (1353), SoundexCompare (1354), SoundexInt (1354), SoundexProc (1354), SoundexWord (1355), Soundex (1353)

74.3.94 SoundexWord

Synopsis: Calculate a word-sized soundex value.

Declaration:
```
function SoundexWord(const AText: string) : Word
```

Visibility: default

Description: `SoundexInt` computes the Soundex (1353) code (with length 4) of `AText`, and converts the code to a word-sized value.

`AText` may not contain null characters.

Errors: None.

See also: Soundex (1353), SoundexCompare (1354), SoundexInt (1354), SoundexProc (1354), SoundexSimilar (1355)

74.3.95 SplitCommandLine

Synopsis: Split commandline in a series of separate arguments.

Declaration:
```
function SplitCommandLine(S: RawByteString) : TRawByteStringArray
function SplitCommandLine(S: UnicodeString) : TUnicodeStringArray
```

Visibility: default

Description: `SplitCommandline` parses `S` into words separated by whitespace, and returns a list of separate words (command-line arguments): When parsing it takes into account quoting with single (') or double quote (") characters: a word that contains whitespace can be enclosed in a quote character and will be considered a single word.

You can call this function with a `UnicodeString` or a `RawByteString`, the resulting array will have the same type of elements.

for example:

```
first "second argument" 'third argument'
```

Will result in an array of 3 elements

- first
- second argument
- thirs argument

See also: TRawByteStringArray (1324), TUnicodeStringArray (1326)

74.3.96 SplitString

Synopsis: Split a string in words.

Declaration:
```
function SplitString(const S: string; const Delimiters: string)
                           : TStringDynArray
```

Visibility: default

Description: SplitString will split the string (S) using the characters in Delimiters as separator characters. The result contains all words separated by one of the characters in Delimiters.

This function is a simplified wrapper around TStringHelper.Split (1323).

See also: TStringHelper.Split (1323)

74.3.97 StartsStr

Synopsis: Check whether one string starts with another.

Declaration:
```
function StartsStr(const ASubText: string; const AText: string)
                         : Boolean
```

Visibility: default

Description: StarsText checks whether aText starts with aSubText and returns True if it does. i.e. it returns true if the first characters of aText are aSubText. It follows that the length of aText must be at least the length of aSubText. The comparison is made case-sensitive. If you wish to compare case-insentively, use StartsText (1356) instead.

See also: AnsiStartsStr (1331), EndsStr (1336), StartsText (1356)

74.3.98 StartsText

Synopsis: Check whether one text starts with another.

Declaration:
```
function StartsText(const ASubText: string; const AText: string)
                           : Boolean
```

Visibility: default

Description: StarsText checks whether aText starts with aSubText and returns True if it does. i.e. it returns true if the first characters of aText are aSubText. It follows that the length of aText must be at least the length of aSubText. The comparison is made case-insensitive. If you wish to compare case-sensitively, use StartsStr (1356) instead.

See also: AnsiStartsText (1331), EndsText (1336), StartsStr (1356)

74.3.99 StringReplace

Synopsis: Optimized search-and-replace algorithm.

Declaration:
```
function StringReplace(const S: string; const OldPattern: string;
                       const NewPattern: string; Flags: TReplaceFlags;
                       out aCount: Integer;
                       Algorithm: TStringReplaceAlgorithm) : string
                       ;  Overload
function StringReplace(const S: string; const OldPattern: string;
                       const NewPattern: string; Flags: TReplaceFlags;
                       Algorithm: TStringReplaceAlgorithm) : string
                       ;  Overload
function StringReplace(const S: UnicodeString;
                       const OldPattern: UnicodeString;
                       const NewPattern: UnicodeString;
                       Flags: TReplaceFlags) : UnicodeString;  Overload
function StringReplace(const S: WideString;
                       const OldPattern: WideString;
                       const NewPattern: WideString;
                       Flags: TReplaceFlags) : WideString;  Overload
```

Visibility: default

Description: `StringReplace` replaces one or all occurrences of `OldPattern` with `NewPattern` in the string `S`. The behaviour is controlled by `Flags` and the search mechanism may be optimized using `Algorithm`. A list of possible algorithms is specified in TStringReplaceAlgorithm (1325).

For ansistrings, this is an optimized version of the SysUtils.StringReplace (1778) algorithm. For widestrings or unicodestrings, the algorithm parameter is ignored and the default mechanism in sysutils is always used.

Errors: None.

See also: SysUtils.StringReplace (1778), TStringReplaceAlgorithm (1325)

74.3.100 StringsReplace

Synopsis: Replace occurrences of a set of strings to another set of strings.

Declaration:
```
function StringsReplace(const S: string; OldPattern: Array of string;
                        NewPattern: Array of string;
                        Flags: TReplaceFlags) : string
```

Visibility: default

Description: `StringsReplace` scans `S` for the occurrence of one of the strings in `OldPattern` and replaces it with the corresponding string in `NewPattern`. It takes into account `Flags`, which has the same meaning as in StringReplace (1778).

Corresponding strings are matched by location: the N-th string in `OldPattern` is replaced by the N-th string in `NewPattern`. Note that this means that the number of strings in both arrays must be the same.

Errors: If the number of strings in both arrays is different, then an exception is raised.

See also: StringReplace (1778), TReplaceFlags (1659)

74.3.101 StuffString

Synopsis: Replace part of a string with another string.

Declaration:
```
function StuffString(const AText: string; AStart: Cardinal;
                     ALength: Cardinal; const ASubText: string) : string
```

Visibility: default

Description: `StuffString` returns a copy of `AText` with the segment starting at `AStart` with length `ALength`, replaced with the string `ASubText`. Basically it deletes the segment of `Atext` and inserts the new text in it's place.

Errors: No checking on the validity of the AStart and ALength parameters is done. Providing invalid values may result in access violation errors.

See also: FindPart (1339), DelChars (1335), DelSpace (1335), ExtractSubStr (1337), DupeString (1335)

74.3.102 Tab2Space

Synopsis: Convert tab characters to a number of spaces.

Declaration:
```
function Tab2Space(const S: string; Numb: Byte) : string
```

Visibility: default

Description: `Tab2Space` returns a copy of `S` with all tab characters (ASCII character 9) converted to `Numb` spaces.

Errors: None.

See also: StuffString (1358), FindPart (1339), ExtractWord (1337), DelChars (1335), DelSpace (1335), DelSpace1 (1335), DupeString (1335)

74.3.103 TrimLeftSet

Synopsis: Remove any leading characters in a set from a string and returns the result.

Declaration:
```
function TrimLeftSet(const S: string; const CSet: TSysCharSet) : string
```

Visibility: default

Description: `TrimLeftSet` performs the same action as RemoveLeadingChars (1349), but returns the resulting string.

Errors: None.

See also: TrimLeft (1803), RemoveLeadingChars (1349), RemoveTrailingChars (1349), RemovePadChars (1349), TrimSet (1359), TrimRightSet (1358)

74.3.104 TrimRightSet

Synopsis: Remove any trailing characters in a set from a string and returns the result.

Declaration:
```
function TrimRightSet(const S: string; const CSet: TSysCharSet) : string
```

Visibility: default

Description: `TrimLeftSet` performs the same action as RemoveTrailingChars (1349), but returns the resulting string.

Errors: None.

See also: TrimRight (1804), RemoveLeadingChars (1349), RemoveTrailingChars (1349), RemovePadChars (1349), TrimSet (1359), TrimLeftSet (1358)

74.3.105 TrimSet

Synopsis: Remove any leading or trailing characters in a set from a string and returns the result.

Declaration:
```
function TrimSet(const S: string; const CSet: TSysCharSet) : string
```

Visibility: default

Description: `TrimSet` performs the same action as RemovePadChars (1349), but returns the resulting string.

Errors: None.

See also: Trim (1803), RemoveLeadingChars (1349), RemoveTrailingChars (1349), RemovePadChars (1349), TrimRightSet (1358), TrimLeftSet (1358)

74.3.106 TryRomanToInt

Synopsis: Try to convert a roman numeral to an integer value.

Declaration:
```
function TryRomanToInt(S: string; out N: LongInt;
                       Strictness: TRomanConversionStrictness) : Boolean
```

Visibility: default

Description: `TryRomanToInt` will try to convert the roman numeral in `S` to an integer and returns the integer value in `N`. The strictness of the conversion algorithm is determined by `Strictness`. If the conversion succeeds, then `True` is returned, or else `False`.

See also: TRomanConversionStrictness (1324), RomanToIntDef (1352), RomanToInt (1351), IntToRoman (1342)

74.3.107 WordCount

Synopsis: Count the number of words in a string.

Declaration:
```
function WordCount(const S: string; const WordDelims: TSysCharSet)
                   : SizeInt
```

Visibility: default

Description: `WordCount` returns the number of words in the string `S`. A word is a non-empty string of characters bounded by one of the characters in `WordDelims`.

The predefined StdWordDelims (1324) constant can be used for the `WordDelims` argument.

Errors: None.

See also: WordPosition (1360), StdWordDelims (1324), ExtractWord (1337), ExtractWordPos (1337)

74.3.108 WordPosition

Synopsis: Search position of Nth word in a string.

Declaration:
```
function WordPosition(const N: Integer; const S: string;
                      const WordDelims: TSysCharSet) : SizeInt
```

Visibility: default

Description: `WordPosition` returns the position (in characters) of the `N`-th word in the string `S`. A word is a non-empty string of characters bounded by one of the characters in `WordDelims`. If `N` is out of range, zero is returned.

The predefined StdWordDelims (1324) constant can be used for the `WordDelims` argument.

Errors: None

See also: WordCount (1359), StdWordDelims (1324), ExtractWord (1337), ExtractWordPos (1337)

74.3.109 XorDecode

Synopsis: Decode a string encoded with XorEncode (1360).

Declaration:
```
function XorDecode(const Key: Ansistring; const Source: Ansistring)
                   : Ansistring
```

Visibility: default

Description: `XorDecode` decodes `Source` and returns the original string that was encrypted using XorEncode (1360) with key `Key`. If a different key is used than the key used to encode the string, the result will be unreadable.

Errors: If the string `Source` is not a valid XorEncode result (e.g. contains non-numerical characters), then a `EConversionError` exception will be raised.

See also: XorEncode (1360), XorString (1361)

74.3.110 XorEncode

Synopsis: Encode a string by XOR-ing its characters using characters of a given key, representing the result as hex values.

Declaration:
```
function XorEncode(const Key: Ansistring; const Source: Ansistring)
                   : Ansistring
```

Visibility: default

Description: `XorEncode` encodes the string `Source` by XOR-ing each character in `Source` with the corresponding character in `Key` (repeating `Key` as often as necessary) and representing the resulting ASCII code as a hexadecimal number (of length 2). The result is therefore twice as long as the original string, and every 2 bytes represent an ASCII code.

Feeding the resulting string with the same key `Key` to the XorDecode (1360) function will result in the original `Source` string.

This function can be used e.g. to trivially encode a password in a configuration file.

Errors: None.

See also: XorDecode (1360), XorString (1361), Hex2Dec (1340)

74.3.111 XorString

Synopsis: Encode a string by XOR-ing its characters using characters of a given key.

Declaration:
```
function XorString(const Key: ShortString; const Src: ShortString)
                         : ShortString
```

Visibility: default

Description: `XorString` encodes the string `Src` by XOR-ing each character in `Source` with the corresponding character in `Key`, repeating `Key` as often as necessary. The resulting string may contain unreadable characters and may even contain null characters. For this reason it may be a better idea to use the XorEncode (1360) function instead, which will representing each resulting ASCII code as a hexadecimal number (of length 2).

Feeding the result again to `XorString` with the same `Key`, will result in the original string `Src`.

Errors: None.

See also: XorEncode (1360), XorDecode (1360)

Chapter 75

Reference for unit 'System'

75.1 Overview

The system unit contains the standard supported functions of Free Pascal. It is the same for all platforms. Basically it is the same as the system unit provided with Borland or Turbo Pascal.

Functions are listed in alphabetical order. Arguments of functions or procedures that are optional are put between square brackets.

The predefined constants and variables are listed in the first section. The second section contains an overview of all functions, grouped by functionality, and the last section contains the supported functions and procedures.

Remark The RTL documentation is generated from the actual sources sources. This means that for some CPU-dependent or platform-dependent types, the shown definition depends on the platform and CPU for which the documentation is generated. (which is normally the Linux platform and i386 CPU)

75.2 A list of managed types.

Some of the basic pascal types are managed. This means that their lifetime is treated specially by the compiler: Variables of a managed type:

- Are implicitly initialized at the beginning of their scope. Usually this means the memory area reserved for them is zeroed out.
- Use some form of reference counting when they are assigned to one another during their lifetime.
- Are implicitly finalized at the end of their scope. Usually this means decreasing a reference count and freeing the dynamic memory occupied by the variable if the reference count returns zero.

The following types are managed:

Ansistring The one-byte character string uses a reference count.

UnicodeString The two-byte character string uses a reference count

Widestring Is equal to Unicodestring on non-windows platforms, is managed specially on windows.

Dynamic arrays Are reference counted.

Interfaces COM interfaces are reference counted through the mechanisms in `IInterface`. CORBA interfaces are not managed.

Additionally, structured types such as arrays, records, objects and classes containing one of the above types automatically become managed too. Note that for classes this does not mean that the class instance itself becomes managed, but when creating/destroying the class through the constructor and destructor, the fields in a class that are of managed type will be initialized and finalized.

For more information about the exact workings for each type, please consult the language reference guide.

75.3 Unicode and codepage support.

The system unit works with Short-, Ansi-, and UnicodeString routines for all string related operations.

Ansistrings are code-page aware, which means that code page information is associated with them. For most routines, the support for converting these code pages is natural. For some routines, care must be taken when converting from codepage-aware strings to widestring.

The codepage conversion support is influenced by the following variables:

Table 75.1:

Name	Description
DefaultSystemCodePage (1443)	Actual code page to use when CP_ACP (1370) is encountered
DefaultUnicodeCodePage (1443)	Code page for new Unicode strings
DefaultFileSystemCodePage (1442)	Codepage to use when sending strings to single-byte OS file system routines.
DefaultRTLFileSystemCodePage (1442)	Codepage to use when receiving strings from single-byte OS file system routines.

The windows code page identifiers are used. There are 3 special codepage identifiers:

Table 75.2:

Name	Description
CP_ACP (1370)	Currently set default system codepage
CP_OEMCP (1370)	OEM (console) code page (only on windows)
CP_NONE (1370)	Indicates absence of code page information for a string
DefaultRTLFileSystemCodePage (1442)	

The following routines may perform code page conversions:

Table 75.3:

Name	Description
LowerCase (1534)	Return lowercase version of a string.
UpCase (1595)	Convert a string to all uppercase.
GetDir (1501)	Return the current directory.
MkDir (1535)	Create a new directory.
ChDir (1464)	Change current working directory.
RmDir (1554)	Remove directory when empty.
Assign (1451)	Assign a name to a file.
Erase (1486)	Delete a file from disk.
Rename (1552)	Rename file on disk.
Read (1547)	Read from a text file into variable.
ReadLn (1549)	Read from a text file into variable and goto next line.
Write (1602)	Write variable to a text file or standard output.
WriteLn (1603)	Write variable to a text file or standard output and append newline.
ReadStr (1549)	Read variables from a string.
WriteStr (1604)	Write variables to a string.
Insert (1521)	Insert one string or dynamic array in another.
Copy (1473)	Copy part of a string.
Delete (1477)	Delete elements (characters) from a string or dynamic array.
SetString (1569)	Set length of a string and copy buffer.

All these routines exist also in Unicode versions.

Note that for conversion of codepages and Unicode strings, a Unicode manager must be present. On windows, the system is used for this. On Unix, one of the fpwidestring or cwstring units must be used.

75.4 Miscellaneous functions.

Functions that do not belong in one of the other categories.

Table 75.4:

Name	Description
Assert (1451)	Conditionally abort program with error
Break (1460)	Abort current loop
Continue (1472)	Next cycle in current loop
Exclude (1487)	Exclude an element from a set
Exit (1488)	Exit current function or procedure
Include (1513)	Include an element into a set
LongJmp (1533)	Jump to execution point
Ord (1540)	Return ordinal value of enumerated type
Pred (1545)	Return previous value of ordinal type
SetJmp (1565)	Mark execution point for jump
SizeOf (1574)	Return size of variable or type
Succ (1582)	Return next value of ordinal type

75.5 Operating System functions.

Functions that are connected to the operating system.

Table 75.5:

Name	Description
Chdir (1464)	Change working directory
Getdir (1501)	Return current working directory
Halt (1509)	Halt program execution
Paramcount (1541)	Number of parameters with which program was called
Paramstr (1542)	Retrieve parameters with which program was called
Mkdir (1535)	Make a directory
Rmdir (1554)	Remove a directory
Runerror (1560)	Abort program execution with error condition

75.6 String handling.

All things connected to string handling.

Table 75.6:

Name	Description
BinStr (1458)	Construct binary representation of integer
Chr (1465)	Convert ASCII code to character
Concat (1471)	Concatenate two strings
Copy (1473)	Copy part of a string
Delete (1477)	Delete part of a string
HexStr (1509)	Construct hexadecimal representation of integer
Insert (1521)	Insert one string in another
Length (1528)	Return length of string
Lowercase (1534)	Convert string to all-lowercase
OctStr (1539)	Construct octal representation of integer
Pos (1543)	Calculate position of one string in another
SetLength (1566)	Set length of a string
SetString (1569)	Set contents and length of a string or dynamic array
Str (1577)	Convert number to string representation
StringOfChar (1580)	Create string consisting of a number of characters
Upcase (1595)	Convert string to all-uppercase
Val (1598)	Convert string to number

75.7 Mathematical routines.

Functions connected to calculating and converting numbers.

Table 75.7:

Name	Description
Abs (1446)	Calculate absolute value
Arctan (1450)	Calculate inverse tangent
Cos (1474)	Calculate cosine of angle
Dec (1475)	Decrease value of variable
Exp (1490)	Exponentiate
Frac (1499)	Return fractional part of floating point value
Hi (1510)	Return high byte/word of value
Inc (1512)	Increase value of variable
Int (1521)	Calculate integer part of floating point value
Ln (1530)	Calculate logarithm
Lo (1530)	Return low byte/word of value
Odd (1539)	Is a value odd or even ?
Pi (1542)	Return the value of pi
Random (1546)	Generate random number
Randomize (1547)	Initialize random number generator
Round (1557)	Round floating point value to nearest integer number
Sin (1573)	Calculate sine of angle
Sqr (1576)	Calculate the square of a value
Sqrt (1576)	Calculate the square root of a value
Swap (1583)	Swap high and low bytes/words of a variable
Trunc (1589)	Truncate a floating point value

75.8 Memory management functions.

Functions concerning memory issues.

Table 75.8:

Name	Description
Addr (1448)	Return address of variable
Assigned (1456)	Check if a pointer is valid
CompareByte (1466)	Compare 2 memory buffers byte per byte
CompareChar (1467)	Compare 2 memory buffers byte per byte
CompareDWord (1469)	Compare 2 memory buffers byte per byte
CompareWord (1470)	Compare 2 memory buffers byte per byte
CSeg (1475)	Return code segment
Dispose (1478)	Free dynamically allocated memory
DSeg (1480)	Return data segment
FillByte (1493)	Fill memory region with 8-bit pattern
FillChar (1493)	Fill memory region with certain character
FillDWord (1494)	Fill memory region with 32-bit pattern
FillQWord (**??**)	Fill memory region with 64-bit pattern
FillWord (1494)	Fill memory region with 16-bit pattern
Freemem (1499)	Release allocated memory
Getmem (1502)	Allocate new memory
GetMemoryManager (1503)	Return current memory manager
High (1511)	Return highest index of open array or enumerated
IndexByte (1513)	Find byte-sized value in a memory range
IndexChar (1514)	Find char-sized value in a memory range
IndexDWord (1515)	Find DWord-sized (32-bit) value in a memory range
IndexQWord (1516)	Find QWord-sized value in a memory range
IndexWord (1516)	Find word-sized value in a memory range
IsMemoryManagerSet (1527)	Is the memory manager set
MemSize (1535)	Get size of allocation
Low (1534)	Return lowest index of open array or enumerated
Move (1536)	Move data from one location in memory to another
MoveChar0 (1536)	Move data till first zero character
New (1537)	Dynamically allocate memory for variable
Ofs (1540)	Return offset of variable
Ptr (1545)	Combine segment and offset to pointer
ReAllocMem (1551)	Resize a memory block on the heap
Seg (1564)	Return segment
SetMemoryManager (1567)	Set a memory manager
Sptr (1575)	Return current stack pointer
SSeg (1577)	Return stack segment register value

75.9 File handling functions.

Functions concerning input and output from and to file.

Table 75.9:

Name	Description
Append (1449)	Open a file in append mode
Assign (1451)	Assign a name to a file
Blockread (1459)	Read data from a file into memory
Blockwrite (1460)	Write data from memory to a file
Close (1465)	Close a file
Eof (1484)	Check for end of file
Eoln (1485)	Check for end of line
Erase (1486)	Delete file from disk
Filepos (1491)	Position in file
Filesize (1492)	Size of file
Flush (1497)	Write file buffers to disk
IOresult (1525)	Return result of last file IO operation
Read (1547)	Read from file into variable
Readln (1549)	Read from file into variable and goto next line
Rename (1552)	Rename file on disk
Reset (1552)	Open file for reading
Rewrite (1553)	Open file for writing
Seek (1562)	Set file position
SeekEof (1563)	Set file position to end of file
SeekEoln (1564)	Set file position to end of line
SetTextBuf (1570)	Set size of file buffer
Truncate (1590)	Truncate the file at position
Write (1602)	Write variable to file
WriteLn (1603)	Write variable to file and append newline

75.10 Run-Time Error behaviour.

The system unit handles errors by default by generating a run-time error, and halting the program with an exit code equal to the run-time error number.

This behaviour changes when the SysUtils (1635) unit is used. In that case, all run-time errors are converted to exceptions: most run-time errors have their own exception class.

If these exceptions are caught, the program code decides what to do with it. If the exception is not caught, the program will exit through the default exception handler.

When the system unit documentation refers to run-time errors, the above should be kept in mind.

75.11 Constants, types and variables

75.11.1 Constants

```
AbstractErrorProc : TAbstractErrorProc = Nil
```

If set, the `AbstractErrorProc` constant is used when an abstract error occurs. If it is not set, then the standard error handling is done: A stack dump is performed, and the program exits with error code 211.

The SysUtils unit sets this procedure and raises an exception in its handler.

```
AllFilesMask = '*'
```

AllFilesMask is the wildcard that can be used to return all files in a directory. Do not assume that this is '*' or '*.*' based on the platform only. The actual value on DOS/Windows based systems can be influenced by e.g. LFNSupport (1385).

```
AllowDirectorySeparators : Set of Char = ['\', '/']
```

AllowDirectorySeparators is the set of characters which are considered directory separators by the RTL units. By default, this is set to the most common directory separators: forward slash and backslash, so routines will work in a cross-platform manner, no matter which character was used:

```
AllowDirectorySeparators : set of char = ['\','/'];
```

If a more strict behaviour is desired, then AllowDirectorySeparators can be set to the only character allowed on the current operating system, and all RTL routines that handle filenames (splitting filenames, extracting parts of the filename and so on) will use that character only.

```
AllowDriveSeparators : Set of Char = []
```

AllowDriveSeparators are the characters which are considered to separate the drive part from the directory part in a filename. This will be an empty set on systems that do not support drive letters. Other systems (Dos, Windows and OS/2) will have the colon (:) character as the only member of this set.

```
AssertErrorProc : TAssertErrorProc = @ SysAssert
```

If set, the AssertErrorProc constant is used when an assert error occurs. If it is not set, then the standard assert error handling is performed:

- The assertion error message is printed.
- The location of the assertion is printed.
- a stack dump is printed.
- The program exits with error code 227.

The SysUtils unit sets this procedure and raises an exception in its handler.

```
BacktraceStrFunc : TBacktraceStrFunc = @ SysBacktraceStr
```

This handler is called to get a standard format for the backtrace routine.

```
CatchAllExceptions = PtrInt(- 1)
```

Mask indicating all exception kinds.

```
cExceptionFrame = 1
```

cExceptionFrame indicates an except frame.

```
cFinalizeFrame = 2
```

cFinalizeFrame indicates a finally frame.

```
CP_ACP = 0
```

CP_ACP is the default Windows codepage identifier.

```
CP_ASCII = 20127
```

CP_ASCII is the Windows ASCII encoding codepage identifier.

```
CP_NONE = $FFFF
```

CP_NONE is used when no code page information is available.

```
CP_OEMCP = 1
```

CP_ACP is the default Windows OEM (MS-DOS) codepage identifier.

```
CP_UTF16 = 1200
```

CP_UTF16 is the default Windows Unicode codepage identifier (little endian).

```
CP_UTF16BE = 1201
```

CP_UTF16BE is the Windows Unicode codepage identifier (big endian).

```
CP_UTF7 = 65000
```

CP_UTF7 is the Windows Unicode 7-Bit encoding codepage identifier.

```
CP_UTF8 = 65001
```

CP_UTF8 is the Windows Unicode 8-Bit encoding codepage identifier.

```
CtrlZMarksEOF : Boolean = False
```

CtrlZMarksEOF indicates whether on this system, an CTRL-Z character (ordinal 26) in a file marks the end of the file. This is False on most systems apart from DOS and Windows.
To get DOS/Windows-compatible behaviour, this constant can be set to True

```
Default8087CW : Word = $1332
```

```
DefaultMXCSR : DWord = $1900
```

```
DefaultStackSize = 4 * 1024 * 1024
```

Default size for a new thread's stack (4MiB by default).

```
DefaultTextLineBreakStyle : TTextLineBreakStyle = tlbsLF
```

DefaultTextLineBreakStyle contains the default OS setting for the TTextLineBreakStyle (1427) type. It is initialized by the system unit, and is used to determine the default line ending when writing to text files.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
DirectorySeparator = '/'
```

DirectorySeparator is the character used by the current operating system to separate directory parts in a pathname. This constant is system dependent, and should not be set.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
DriveSeparator = ''
```

On systems that support drive letters, the DriveSeparator constant denotes the character that separates the drive indicator from the directory part in a filename path.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
ErrorAddr : CodePointer = Nil
```

Address where the last error occurred.

```
ErrorCode : Word = 0
```

Last error code.

```
ErrorProc : TErrorProc = Nil
```

If set, the ErrorProc constant is used when a run-time error occurs. If it is not set, then the standard error handling is done: a stack dump is performed, and the program exits with the indicated error code.

The SysUtils unit sets this procedure and raises an exception in its handler.

```
ExceptClsProc : Pointer = Nil
```

ExceptClsProc is used in SEH (Structured Exception Support) to convert OS exception information to FPC exception classes when filtering exceptions. It is set e.g. by the sysutils unit. If it is not set, the exception is not handled.

```
ExceptObjProc : Pointer = Nil
```

ExceptObjProc is used in SEH (Structured Exception Support) to convert OS exception information to FPC exceptions. It is set e.g. by the sysutils unit. If it is not set, a run-time error results when OS exceptions are intercepted.

```
ExceptProc : TExceptProc = Nil
```

This constant points to the current exception handling procedure. This routine is called when an unhandled exception occurs, i.e. an exception that is not stopped by a except block.

If the handler is not set, the RTL will emit a run-time error 217 when an unhandler exception occurs.

It is set by the sysutils (1635) unit.

```
ExitProc : CodePointer = Nil
```

Exit procedure pointer.

```
ExtensionSeparator = '.'
```

`ExtensionSeparator` is the character which separates the filename from the file extension. On all current platforms, this is the `.` (dot) character. All RTL filename handling routines use this constant.

```
E_NOINTERFACE = hresult($80004002)
```

Interface call result: Error: not an interface.

```
E_NOTIMPL = hresult($80004001)
```

Interface call result: Interface not implemented.

```
E_UNEXPECTED = hresult($8000FFFF)
```

Interface call result: Unexpected error.

```
FileMode : Byte = 2
```

`FileMode` determines how untyped files are opened for reading with Reset (1552). It can have the following values:

0 open file readonly

1 open file write only

2 open file read/write

```
FileNameCasePreserving : Boolean = True
```

`FileNameCasePreserving` is `True` if case of letters in file and directory entries is preserved and may be later retrieved exactly as supplied when creating or renaming these entries. Note that this may depend on the file system: Unix operating systems that access a DOS or Windows partition will have this constant set to true, but when writing to the DOS partition, all letters may be automatically converted to uppercase.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
FileNameCaseSensitive : Boolean = True
```

`FileNameCaseSensitive` is `True` if case is important when using filenames on the current OS. In this case, the OS will treat files with different cased names as different files. Note that this may depend on the file system: Unix operating systems that access a DOS or Windows partition will have this constant set to true, but when writing to the DOS partition, the casing is ignored.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
filerecnamelength = 255
```

`filerecnamelength` is the maximum filename size for untyped files.

```
float_flag_denormal = exDenormalized
```

IEC/IEEE floating-point exception flag: ?

```
float_flag_divbyzero = exZeroDivide
```

IEC/IEEE floating-point exception flag: Division by zero error.

```
float_flag_inexact = exPrecision
```

IEC/IEEE floating-point exception flag: ?

```
float_flag_invalid = exInvalidOp
```

IEC/IEEE floating-point exception flag: Invalid operation error.

```
float_flag_overflow = exOverflow
```

IEC/IEEE floating-point exception flag: Overflow error.

```
float_flag_underflow = exUnderflow
```

IEC/IEEE floating-point exception flag: Underflow error.

```
float_round_down = rmDown
```

Round down.

```
float_round_nearest_even = rmNearest
```

Round to nearest even number.

```
float_round_to_zero = rmTruncate
```

Round in the direction of zero (down for positive, up for negative).

```
float_round_up = rmUp
```

Round up.

```
fmAppend = $D7B4
```

File mode: File is open for writing, appending to the end.

```
fmClosed = $D7B0
```

File mode: File is closed.

```
fmInOut = $D7B3
```

File mode: File is open for reading and writing.

```
fmInput = $D7B1
```

File mode: File is open for reading.

```
fmOutput = $D7B2
```

File mode: File is open for writing.

```
FPC_EXCEPTION = 1
```

`FPC_EXCEPTION` is a contstant describing a type of excption.

```
fpc_in_abs_long = 64
```

Internal ABS function.

```
fpc_in_abs_real = 127
```

FPC compiler internal procedure index: abs (real).

```
fpc_in_addr_x = 42
```

FPC compiler internal procedure index: addr.

```
fpc_in_aligned_x = 80
```

FPC compiler internal procedure index: aligned.

```
fpc_in_and_assign_x_y = 86
```

```
fpc_in_arctan_real = 130
```

FPC compiler internal procedure index: arctan (real).

```
fpc_in_assert_x_y = 41
```

FPC compiler internal procedure index: assert.

```
fpc_in_assigned_x = 19
```

FPC compiler internal procedure index: assigned.

```
fpc_in_bitsizeof_x = 61
```

FPC compiler internal procedure index: bitsizeof.

```
fpc_in_box_x = 77
```

FPC compiler internal procedure index: box.

```
fpc_in_break = 39
```

FPC compiler internal procedure index: break.

```
fpc_in_bsf_x = 74
```

FPC compiler internal procedure index: bsf_x.

```
fpc_in_bsr_x = 75
```

FPC compiler internal procedure index: bsr_x.

```
fpc_in_chr_byte = 7
```

FPC compiler internal procedure index: chr.

```
fpc_in_concat_x = 18
```

FPC compiler internal procedure index: concat.

```
fpc_in_const_abs = 101
```

FPC compiler internal procedure index: abs.

```
fpc_in_const_odd = 102
```

FPC compiler internal procedure index: sqr.

```
fpc_in_const_ptr = 103
```

FPC compiler internal procedure index: sqr.

```
fpc_in_const_sqr = 100
```

FPC compiler internal procedure index: sqr.

```
fpc_in_const_swap_long = 105
```

FPC compiler internal procedure index: swap (long).

```
fpc_in_const_swap_qword = 108
```

FPC compiler internal procedure index: swap (qword).

```
fpc_in_const_swap_word = 104
```

FPC compiler internal procedure index: swap (word).

```
fpc_in_continue = 40
```

FPC compiler internal procedure index: continue.

```
fpc_in_copy_x = 49
```

FPC compiler internal procedure index: copy.

```
fpc_in_cos_real = 125
```

FPC compiler internal procedure index: cos (real).

```
fpc_in_cpu_first = 10000
```

```
fpc_in_cycle = 52
```

FPC compiler internal procedure index: cycle.

```
fpc_in_dec_x = 36
```

FPC compiler internal procedure index: dec.

```
fpc_in_default_x = 76
```

FPC compiler internal procedure index: default.

```
fpc_in_delete_x_y_z = 83
```

```
fpc_in_dispose_x = 47
```

FPC compiler internal procedure index: dispose.

```
fpc_in_exclude_x_y = 38
```

FPC compiler internal procedure index: exclude.

```
fpc_in_exit = 48
```

FPC compiler internal procedure index: exit.

```
fpc_in_exp_real = 124
```

FPC internal compiler routine: in_exp_real.

```
fpc_in_faraddr_x = 97
```

```
fpc_in_fillchar_x = 55
```

FPC internal compiler routine: in_fillchar_x.

```
fpc_in_finalize_x = 45
```

FPC compiler internal procedure index: finalize.

```
fpc_in_fma_double = 134
```

FPC compiler internal procedure index: fma (double).

```
fpc_in_fma_extended = 135
```

FPC compiler internal procedure index: fma (extended).

```
fpc_in_fma_float128 = 136
```

FPC compiler internal procedure index: fma (float 128).

```
fpc_in_fma_single = 133
```

FPC compiler internal procedure index: fma (single).

```
fpc_in_frac_real = 122
```

FPC internal compiler routine: in_frac_real.

```
fpc_in_get_caller_addr = 57
```

FPC internal compiler routine: in_get_caller_addr.

```
fpc_in_get_caller_frame = 58
```

FPC internal compiler routine: in_get_caller_frame.

```
fpc_in_get_frame = 56
```

FPC internal compiler routine: in_get_frame.

```
fpc_in_high_x = 28
```

FPC compiler internal procedure index: high.

```
fpc_in_hi_long = 4
```

FPC compiler internal procedure index: hi (long).

```
fpc_in_hi_qword = 107
```

FPC compiler internal procedure index: hi (qword).

```
fpc_in_hi_word = 2
```

FPC compiler internal procedure index: hi (word).

```
fpc_in_include_x_y = 37
```

FPC compiler internal procedure index: include.

```
fpc_in_inc_x = 35
```

FPC compiler internal procedure index: inc.

```
fpc_in_initialize_x = 50
```

FPC compiler internal procedure index: initialize.

```
fpc_in_insert_x_y_z = 82
```

```
fpc_in_int_real = 123
```

FPC internal compiler routine: in_int_real.

```
fpc_in_leave = 51
```

FPC compiler internal procedure index: leave.

```
fpc_in_length_string = 6
```

FPC compiler internal procedure index: length.

```
fpc_in_ln_real = 131
```

FPC compiler internal procedure index: ln (real).

```
fpc_in_low_x = 27
```

FPC compiler internal procedure index: low.

```
fpc_in_lo_long = 3
```

FPC compiler internal procedure index: lo (long).

```
fpc_in_lo_qword = 106
```

FPC compiler internal procedure index: lo (qword).

```
fpc_in_lo_word = 1
```

FPC compiler internal procedure index: lo (word).

```
fpc_in_mmx_pcmpeqb = 200
```

FPC compiler internal procedure index: MMX.

```
fpc_in_mmx_pcmpeqd = 202
```

FPC compiler internal procedure index: MMX.

```
fpc_in_mmx_pcmpeqw = 201
```

FPC compiler internal procedure index: MMX.

```
fpc_in_mmx_pcmpgtb = 203
```

FPC compiler internal procedure index: MMX.

```
fpc_in_mmx_pcmpgtd = 205
```

FPC compiler internal procedure index: MMX.

```
fpc_in_mmx_pcmpgtw = 204
```

FPC compiler internal procedure index: MMX.

```
fpc_in_move_x = 54
```

FPC internal compiler routine: in_move_x.

```
fpc_in_neg_assign_x = 94
```

```
fpc_in_new_x = 46
```

FPC compiler internal procedure index: new.

```
fpc_in_not_assign_x = 95
```

```
fpc_in_ofs_x = 21
```

FPC compiler internal procedure index: ofs.

```
fpc_in_ord_x = 5
```

FPC compiler internal procedure index: ord.

```
fpc_in_or_assign_x_y = 87
```

```
fpc_in_pack_x_y_z = 59
```

FPC compiler internal procedure index: pack.

```
fpc_in_pi_real = 126
```

FPC internal compiler routine: in_pi_real.

```
fpc_in_popcnt_x = 79
```

FPC compiler internal procedure index: popcnt.

```
fpc_in_pred_x = 30
```

FPC compiler internal procedure index: pred.

```
fpc_in_prefetch_var = 109
```

FPC compiler internal procedure index: prefetch.

```
fpc_in_readln_x = 17
```

FPC compiler internal procedure index: readln.

```
fpc_in_readstr_x = 63
```

Internal read string procedure.

```
fpc_in_read_x = 16
```

FPC compiler internal procedure index: read.

```
fpc_in_reset_typedfile = 32
```

FPC compiler internal procedure index: reset.

```
fpc_in_reset_typedfile_name = 84
```

```
fpc_in_reset_x = 25
```

FPC compiler internal procedure index: reset.

```
fpc_in_rewrite_typedfile = 33
```

FPC compiler internal procedure index: rewrite.

```
fpc_in_rewrite_typedfile_name = 85
```

```
fpc_in_rewrite_x = 26
```

FPC compiler internal procedure index: rewrite.

```
fpc_in_rol_assign_x_y = 92
```

```
fpc_in_rol_x = 67
```

```
fpc_in_rol_x_x = 68
```

```
fpc_in_ror_assign_x_y = 93
```

```
fpc_in_ror_x = 65
```

```
fpc_in_ror_x_x = 66
```

```
fpc_in_round_real = 121
```

FPC internal compiler routine: in_round_real.

```
fpc_in_sar_assign_x_y = 89
```

```
fpc_in_sar_x = 73
```

FPC compiler internal procedure index: sar_x.

```
fpc_in_sar_x_y = 72
```

FPC compiler internal procedure index: sar_x_y.

```
fpc_in_seg_x = 29
```

FPC compiler internal procedure index: seg.

```
fpc_in_setlength_x = 44
```

FPC compiler internal procedure index: setlength.

```
fpc_in_setstring_x_y_z = 81
```

FPC compiler internal procedure index: setstring.

```
fpc_in_settextbuf_file_x = 34
```

FPC compiler internal procedure index: settextbuf.

```
fpc_in_shl_assign_x_y = 90
```

```
fpc_in_shr_assign_x_y = 91
```

```
fpc_in_sin_real = 132
```

FPC compiler internal procedure index: sin (real).

```
fpc_in_sizeof_x = 22
```

FPC compiler internal procedure index: sizeof.

```
fpc_in_slice = 53
```

FPC internal compiler routine: in_slice.

```
fpc_in_sqrt_real = 129
```

FPC compiler internal procedure index: sqrt (real).

```
fpc_in_sqr_real = 128
```

FPC compiler internal procedure index: sqr (real).

```
fpc_in_str_x_string = 20
```

FPC compiler internal procedure index: str.

```
fpc_in_succ_x = 31
```

FPC compiler internal procedure index: succ.

```
fpc_in_trunc_real = 120
```

FPC internal compiler routine: in_trunc_real.

```
fpc_in_typeinfo_x = 43
```

FPC compiler internal procedure index: typeinfo.

```
fpc_in_typeof_x = 23
```

FPC compiler internal procedure index: typeof.

```
fpc_in_unbox_x_y = 78
```

FPC compiler internal procedure index: unbox.

```
fpc_in_unpack_x_y_z = 60
```

FPC compiler internal procedure index: unpack.

```
fpc_in_val_x = 24
```

FPC compiler internal procedure index: val.

```
fpc_in_writeln_x = 15
```

FPC compiler internal procedure index: writeln.

```
fpc_in_writestr_x = 62
```

Internal write string procedure.

```
fpc_in_write_x = 14
```

FPC compiler internal procedure index: write.

```
fpc_in_x86_cli = fpc_in_cpu_first + 6
```

```
fpc_in_x86_get_cs = fpc_in_cpu_first + 8
```

```
fpc_in_x86_get_ds = fpc_in_cpu_first + 10
```

```
fpc_in_x86_get_es = fpc_in_cpu_first + 11
```

```
fpc_in_x86_get_fs = fpc_in_cpu_first + 12
```

```
fpc_in_x86_get_gs = fpc_in_cpu_first + 13
```

```
fpc_in_x86_get_ss = fpc_in_cpu_first + 9
```

```
fpc_in_x86_inportb = fpc_in_cpu_first
```

```
fpc_in_x86_inportl = fpc_in_cpu_first + 2
```

```
fpc_in_x86_inportw = fpc_in_cpu_first + 1
```

```
fpc_in_x86_outportb = fpc_in_cpu_first + 3
```

```
fpc_in_x86_outportl = fpc_in_cpu_first + 5
```

```
fpc_in_x86_outportw = fpc_in_cpu_first + 4
```

```
fpc_in_x86_sti = fpc_in_cpu_first + 7
```

```
fpc_in_xor_assign_x_y = 88
```

```
fpc_objc_encode_x = 71
```

```
fpc_objc_protocol_x = 70
```

```
fpc_objc_selector_x = 69
```

```
growheapsize1 : PtrUInt = 256 * 1024
```

Grow rate for block less than 256 Kb.

```
growheapsize2 : PtrUInt = 1024 * 1024
```

Grow rate for block larger than 256 Kb.

```
growheapsizesmall : PtrUInt = 32 * 1024
```

Fixed size small blocks grow rate.

```
has_mmx_support : Boolean = False
```

CPU has mmx support.

```
has_sse2_support : Boolean = False
```

CPU has sse2 support?

```
has_sse3_support : Boolean = False
```

CPU has sse3 support?

```
has_sse_support : Boolean = False
```

CPU has SSE support ?

```
InitProc : CodePointer = Nil
```

`InitProc` is a routine that can be called after all units were initialized. It can be set by units to execute code that can be initialized after all units were initialized.

Remark When setting the value of `InitProc`, the previous value should always be saved, and called when the installed initialization routine has finished executing.

```
IObjectInstance : TGuid =
  '{D91C9AF4-3C93-420F-A303-BF5BA82BFD23}'
```

`IObjectInstance` is an internal GUID, which should not be used in end-user code. It is used in the `as` operator.

```
IsMultiThread : longbool = False
```

Indicates whether more than one thread is running in the application.

```
LFNSupport = True
```

`LFNSupport` determines whether the current OS supports long file names, i.e. filenames that are not of the form 8.3 as on ancient DOS systems. If the value of this constant is `True` then long filenames are supported. If it is false, then not.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
LineEnding = #10
```

`LineEnding` is a constant which contains the current line-ending character. This character is system dependent, and is initialized by the system. It should not be set.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
maxExitCode = 255
```

`maxExitCode` is the maximum value for the Halt (1509) call.

```
MaxInt = maxsmallint
```

Maximum integer value.

```
MaxKeptOSChunks : DWord = 4
```

`MaxKeptOSChunks` tells the heap manager how many free chunks of OS-allocated memory it should keep in memory. When freeing memory, it can happen that a memory block obtained from the OS is completely free. If more than `MaxKeptOSChunks` such blocks are free, then the heap manager will return them to the OS, to reduce memory requirements.

```
MaxLongint = $7fffffff
```

Maximum longint value.

```
MaxPathLen = 4096
```

This constant is system dependent.

```
MaxSIntValue = High(ValSInt)
```

Maximum String-size value.

```
MaxSmallint = 32767
```

Maximum smallint value.

```
MaxUIntValue = High(ValUInt)
```

Maximum unsigned integer value.

```
Max_Frame_Dump : Word = 8
```

Maximum number of frames to show in error frame dump.

```
ModuleIsCpp : Boolean = False
```

`ModuleIsCpp` is always false for FPC programs, it is provided for Delphi compatibility only.

```
ModuleIsLib : Boolean = False
```

`ModuleIsLib` is set by the compiler when linking a library, program or package, and determines whether the current module is a library (or package) (`True`) or program (`False`).

```
ModuleIsPackage : Boolean = False
```

`ModuleIsLib` is set by the compiler when linking a library, program or package, and determines whether the current module is a package (`True`) or a library or program (`False`).

```
NilHandle = TLibHandle(0)
```

Correctly typed Nil handle - returned on error by LoadLibrary (1531).

```
PathSeparator = ':'
```

`PathSeparator` is the character used commonly on the current operating system to separate paths in a list of paths, such as the `PATH` environment variable.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
RaiseMaxFrameCount : LongInt = 16
```

Maximum number of frames to include in TExceptObject (1416).

```
RaiseProc : TExceptProc = Nil
```

Procedure to raise an exception.

```
RT_ACCELERATOR = MAKEINTRESOURCE(9)
```

Constant identifying an accelerator resource.

```
RT_ANICURSOR = MAKEINTRESOURCE(21)
```

This constant can be used to specify a resource of type "animated cursor".

```
RT_ANIICON = MAKEINTRESOURCE(22)
```

This constant can be used to specify a resource of type "animated icon".

```
RT_BITMAP = MAKEINTRESOURCE(2)
```

Constant identifying a bitmap resource.

```
RT_CURSOR = MAKEINTRESOURCE(1)
```

Constant identifying a cursor resource.

```
RT_DIALOG = MAKEINTRESOURCE(5)
```

Constant identifying a dialog resource.

```
RT_FONT = MAKEINTRESOURCE(8)
```

Constant identifying a font resource.

```
RT_FONTDIR = MAKEINTRESOURCE(7)
```

Constant identifying a font directory resource.

```
RT_GROUP_CURSOR = MAKEINTRESOURCE(12)
```

Constant identifying a group cursor resource.

```
RT_GROUP_ICON = MAKEINTRESOURCE(14)
```

Constant identifying a group icon resource.

```
RT_HTML = MAKEINTRESOURCE(23)
```

This constant can be used to specify a resource of type "HTML data".

```
RT_ICON = MAKEINTRESOURCE(3)
```

Constant identifying an icon resource.

```
RT_MANIFEST = MAKEINTRESOURCE(24)
```

This constant can be used to specify a resource of type "Manifest".

```
RT_MENU = MAKEINTRESOURCE(4)
```

Constant identifying a menu resource.

```
RT_MESSAGETABLE = MAKEINTRESOURCE(11)
```

Constant identifying a message data resource.

```
RT_RCDATA = MAKEINTRESOURCE(10)
```

Constant identifying a binary data resource.

```
RT_STRING = MAKEINTRESOURCE(6)
```

Constant identifying a string table resource.

```
RT_VERSION = MAKEINTRESOURCE(16)
```

Constant identifying a version info resource.

```
RuntimeErrorExitCodes : Array[TRuntimeError] of Byte = (0, 203, 204
  , 200, 201, 215, 207, 200, 205, 206, 219, 216, 218, 217, 202, 220
  , 221, 222, 223, 224, 225, 227, 212, 228, 229, 233, 234, 235, 236
  )
```

This array is used by the Error (1487) routine to convert a TRuntimeError (1424) enumeration type to a process exit code.

```
SafeCallErrorProc : TSafeCallErrorProc = Nil
```

`SafeCallErrorProc` is a Handler called in case of a safecall calling convention error. `Error` is the error number (passed by the Windows operating system) and `Addr` is the address where the error occurred.

```
SharedSuffix = 'so'
```

Shared library suffix for the current platform.

```
sLineBreak = LineEnding
```

`sLineBreak` is an alias for LineEnding (1385) and is supplied for Delphi compatibility.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
StackError : Boolean = False
```

Indicate whether there was a stack error.

```
StdErrorHandle = 2
```

Value of the OS handle for the standard error-output file.

```
StdInputHandle = 0
```

Value of the OS handle for the standard input file.

```
StdOutputHandle = 1
```

Value of the OS handle for the standard output file.

```
S_FALSE = 1
```

Interface call result: Not OK.

```
S_OK = 0
```

Interface call result: OK.

```
Test8086 : Byte = 2
```

This constant will only exist on Intel CPU platforms.

```
Test8087 : Byte = 3
```

This constant will only exist on Intel CPU platforms.

```
TextRecBufSize = 256
```

`TextRecBufSize` is the default buffer size for text files. The actual buffer can be set to another size using SetTextBuf (1570).

```
TextRecNameLength = 256
```

`TextRecNameLength` is the maximum filename size for text files.

```
ThreadingAlreadyUsed : Boolean = False
```

Internal constant for the threading system. Don't use.

```
tkAnsiChar = tkChar
```

`tkAnsiChar` is an alias for TTypeKind.tkChar (1429)

```
tkAnsiString = tkAString
```

`tkAnsiString` is an alias for TTypeKind.tkAString (1429)

```
tkShortString = tkSString
```

`tkShortString` is an alias for TTypeKind.tkSString (1429)

```
tkUnicodeString = tkUString
```

`tkUnicodeString` is an alias for TTypeKind.tkUString (1429)

```
tkWideChar = tkWChar
```

`tkWideChar` is an alias for TTypeKind.tkUChar (1429)

```
tkWideString = tkWString
```

tkWideString is an alias for TTypeKind.tkWString (1429)

```
UnixGetModuleByAddrHook : procedure(addr: pointer; var baseaddr: pointer
  ; var filename: string) = Nil
```

UnixGetModuleByAddrHook is used on Unix systems to retrieve a module name based on an address. It is used in the exeinfo (743) unit to map addresses to module (programs or library) names.

```
UnusedHandle = - 1
```

Value indicating an unused file handle (as reported by the OS).

```
VarAddRefProc : procedure(var v: tvardata) = Nil
```

Callback to increase reference count of a variant.

```
varAny = $101
```

Variant type: Any.

```
varArray = $2000
```

Variant type: variant Array.

```
varBoolean = 11
```

Variant type: Boolean type.

```
varByRef = $4000
```

Variant type: By reference.

```
varByte = 17
```

Variant type: Byte (8 bit).

```
VarClearProc : procedure(var v: tvardata) = Nil
```

Callback to clear a variant.

```
VarCopyProc : procedure(var d: tvardata; const s: tvardata) = Nil
```

Callback to copy a variant.

```
varCurrency = 6
```

Variant type: Currency.

```
varDate = 7
```

Variant type: Date.

```
varDecimal = 14
```

Variant type: Decimal (BCD).

```
varDispatch = 9
```

Variant type: dispatch interface.

```
varDouble = 5
```

Variant type: Double float.

```
varEmpty = 0
```

Variant type: `Empty` variant.

```
varError = 10
```

Variant type: Error type.

```
varInt64 = 20
```

Variant type: Integer (64-Bit).

```
varInteger = 3
```

Variant type: Integer (32-bit).

```
varLongWord = 19
```

Variant type: Word (32 bit).

```
varNull = 1
```

Variant type: Null (1538) variant.

```
varOleStr = 8
```

Variant type: OLE string (widestring).

```
varQWord = 21
```

Variant type: Word (64-bit).

```
varRecord = 36
```

Record variant type.

```
varShortInt = 16
```

Variant type: Shortint (16 bit).

```
varSingle = 4
```

Variant type: Single float.

```
varSmallInt = 2
```

Variant type: smallint (8 bit).

```
varStrArg = $48
```

Variant type: String.

```
varString = $100
```

Variant type: String.

```
VarToLStrProc : procedure(var d: AnsiString; const s: tvardata) =
  Nil
```

Callback to convert a variant to a ansistring.

```
VarToWStrProc : procedure(var d: WideString; const s: tvardata) =
  Nil
```

Callback to convert a variant to a widestring.

```
varTypeMask = $fff
```

Variant type: Mask to extract type.

```
varUInt64 = varQWord
```

`varuint64` denotes an unsigned 64-bit value in a variant. It is one of the values found in the `VType` field of the variant record tvardata (1435).

```
varUnknown = 13
```

Variant type: Unknown.

```
varUStrArg = $49
```

`varustrarg` denotes a Unicode string argument in `DispInvoke` call. It will be converted to varustring in a variant.

```
varUString = $102
```

`varustring` denotes a Unicode string value in a variant. It is one of the values found in the `VType` field of the variant record tvardata (1435).

```
varVariant = 12
```

Variant type: Variant (arrays only).

```
varWord = 18
```

Variant type: Word (16 bit).

```
varWord64 = varQWord
```

Variant type: Word (64-bit).

```
vmtAfterConstruction = vmtMethodStart + sizeof(codepointer) * 5
```

VMT Layout: ?

```
vmtAutoTable = vmtParent + sizeof(pointer) * 7
```

VMT layout: ?

```
vmtBeforeDestruction = vmtMethodStart + sizeof(codepointer) * 6
```

VMT Layout: ?

```
vmtClassName = vmtParent + sizeof(pointer)
```

VMT Layout: location of class name.

```
vmtDefaultHandler = vmtMethodStart + sizeof(codepointer) * 4
```

VMT Layout: ?

```
vmtDefaultHandlerStr = vmtMethodStart + sizeof(codepointer) * 7
```

VMT Layout: ?

```
vmtDestroy = vmtMethodStart
```

VMT Layout: Location of destructor pointer.

```
vmtDispatch = vmtMethodStart + sizeof(codepointer) * 8
```

`vmtDispatch` is the offset from the VMT start, in bytes to the dispatch table for a class.The dispatch table is used when dispatching messages in TObject.Dispatch (1630)

```
vmtDispatchStr = vmtMethodStart + sizeof(codepointer) * 9
```

`vmtDispatchStr` is the offset from the VMT start, in bytes to the dispatch table for a class.The dispatch table is used when dispatching messages in TObject.DispatchStr (1631)

```
vmtDynamicTable = vmtParent + sizeof(pointer) * 2
```

VMT Layout: location of dynamic methods table.

```
vmtEquals = vmtMethodStart + sizeof(codepointer) * 10
```

`vmtEquals` contains the offset from the VMT start, of the location of the TObject.Equals (1633) method pointer.

```
vmtFieldTable = vmtParent + sizeof(pointer) * 4
```

VMT Layout: Location of fields table.

```
vmtFreeInstance = vmtMethodStart + sizeof(codepointer) * 2
```

VMT Layout: location of FreeInstance method.

```
vmtGetHashCode = vmtMethodStart + sizeof(codepointer) * 11
```

`vmtGetHashCode` contains the offset from the VMT start, of the location of the TObject.GetHashCode (1633) method pointer.

```
vmtInitTable = vmtParent + sizeof(pointer) * 6
```

VMT Layout: ?

```
vmtInstanceSize = 0
```

VMT Layout: Location of class instance size in VMT.

```
vmtIntfTable = vmtParent + sizeof(pointer) * 8
```

VMT layout: Interface table.

```
vmtMethodStart = vmtParent + sizeof(pointer) * 10
```

VMT layout: start of method table.

```
vmtMethodTable = vmtParent + sizeof(pointer) * 3
```

VMT Layout: Method table start.

```
vmtMsgStrPtr = vmtParent + sizeof(pointer) * 9
```

VMT layout: message strings table.

```
vmtNewInstance = vmtMethodStart + sizeof(codepointer)
```

VMT Layout: location of NewInstance method.

```
vmtParent = sizeof(SizeInt) * 2
```

VMT Layout: location of pointer to parent VMT.

```
vmtSafeCallException = vmtMethodStart + sizeof(codepointer) * 3
```

VMT Layout: ?

```
vmtToString = vmtMethodStart + sizeof(codepointer) * 12
```

`vmtToString` contains the offset from the VMT start, of the location of the TObject.ToString (1634) method pointer.

```
vmtTypeInfo = vmtParent + sizeof(pointer) * 5
```

VMT Layout: Location of class type information.

```
vtAnsiString = 11
```

TVarRec type: Ansistring.

```
vtBoolean = 1
```

TVarRec type: Boolean.

```
vtChar = 2
```

TVarRec type: Char.

```
vtClass = 8
```

TVarRec type: Class type.

```
vtCurrency = 12
```

TVarRec type: Currency.

```
vtExtended = 3
```

TVarRec type: Extended.

```
vtInt64 = 16
```

TVarRec type: Int64 (signed 64-bit integer).

```
vtInteger = 0
```

TVarRec type: Integer.

```
vtInterface = 14
```

TVarRec type: Interface.

```
vtObject = 7
```

TVarRec type: Object instance.

`vtPChar = 6`

TVarRec type: PChar.

`vtPointer = 5`

TVarRec type: pointer.

`vtPWideChar = 10`

TVarRec type: PWideChar.

`vtQWord = 17`

TVarRec type: QWord (unsigned 64-bit integer).

`vtString = 4`

TVarRec type: String.

`vtUnicodeString = 18`

`vtUnicodeString` denotes a Unicode string argument in the array of const. The `TVarRec.VUnicodeString` field will contain the actual value.

`vtVariant = 13`

TVarRec type: Variant.

`vtWideChar = 9`

TVarRec type: Widechar.

`vtWideString = 15`

TVarRec type: WideString.

75.11.2 Types

`AnsiChar = Char`

Alias for 1-byte sized char.

`Byte = 0..255`

An unsigned 8-bits integer.

`Cardinal = LongWord`

An unsigned 32-bits integer.

```
Char = #0..#255
```

`Char` is the basic ANSI character type, with a range of [#0..#255].

```
CodePointer = Pointer
```

`CodePointer` is used in 8/16-bit targets to indicate pointers to code segments. On all other platforms this equals `Pointer`.

```
CodePointer = Pointer
```

`CodePointer` is used in 8/16-bit targets to indicate pointers to code segments. On all other platforms this equals `Pointer`.

```
CodePtrInt = PtrInt
```

`CodePtrInt` is a signed integer with the same size as CodePointer (1397)

```
CodePtrInt = PtrInt
```

`CodePtrInt` is a signed integer with the same size as CodePointer (1397)

```
CodePtrUInt = PtrUInt
```

`CodePtrUInt` is an unsigned integer with the same size as CodePointer (1397)

```
CodePtrUInt = PtrUInt
```

`CodePtrUInt` is an unsigned integer with the same size as CodePointer (1397)

```
Comp = Int64
```

`Comp` is a 64-bit floating point type. See the language reference for more details.

```
DWord = LongWord
```

An unsigned 32-bits integer.

```
EnumResLangProc = function(ModuleHandle: TFPResourceHMODULE;
  ResourceType: PChar; ResourceName: PChar;
  IDLanguage: Word; lParam: PtrInt) :
  LongBool
```

`EnumResNameProc`s used in the EnumResourceLanguages (1483) call. It is called for all languages for a resource of the specified type and name, and is passed the `ModuleHandle`, `ResourceName`, `ResourceName` and `IDLanguage` values for each language encountered for the specified resource. Additionally, the `lParam` parameter from the `EnumResourceLanguages` is passed unaltered.

```
EnumResNameProc = function(ModuleHandle: TFPResourceHMODULE;
  ResourceType: PChar; ResourceName: PChar;
  lParam: PtrInt) : LongBool
```

EnumResNameProcs used in the EnumResourceNames (1484) call. It is called for all resources of the specified type, and is passed the ModuleHandle, ResourceType, ResourceName values for each resource encountered. Additionally, the lParam parameter from the EnumResourceNames is passed unaltered.

```
EnumResTypeProc = function(ModuleHandle: TFPResourceHMODULE;
  ResourceType: PChar; lParam: PtrInt)
  : LongBool
```

EnumResTypeProc is used in the EnumResourceTypes (1484) call. It is called for all resources, and is passed the ModuleHandle, ResourceType values for each resource encountered. Additionally, the lParam parameter from the EnumResourceTypes is passed unaltered.

```
FarPointer = Pointer
```

FarPointer is used in 8/16-bit targets to indicate far pointers (over segments). On all other platforms this equals Pointer.

```
FileRec = record
public
  Handle : THandle;
  Mode : LongInt;
  RecSize
  : SizeInt;
  _private : Array[1..3*SizeOf(SizeInt)+5*SizeOf(pointer
  )] of Byte;
  UserData : Array[1..32] of Byte;
  name : Array[0..filerecnamelength
  ] of TFileTextRecChar;
end
```

FileRec is the underlying type used in untyped files. It should be treated as opaque and never manipulated directly.

```
HGLOBAL = PtrUInt
```

This is an opaque type.

```
HMODULE = PtrUInt
```

This is an opaque type.

```
HRESULT = LongInt
```

32-Bit signed integer.

```
IInterface = IUnknown
```

`IInterface` is the basic interface from which all COM style interfaces descend.

```
Int16 = SmallInt
```

A signed 16-bits integer.

```
Int32 = LongInt
```

A signed 32-bits integer.

```
Int64 = - 9223372036854775808..9223372036854775807
```

`Int64` is a 64-bit, signed integer type, with range [-9223372036854775808..9223372036854775807].

```
Int8 = ShortInt
```

A signed 8-bits integer.

```
Integer = SmallInt
```

The system unit defines `Integer` as a signed 16-bit integer. But when `DELPHI` or `OBJFPC` mode are active, then the objpas unit redefines `Integer` as a 32-bit integer.

Remark Note that due to the way the language modes are implemented, if you refer to `integer` with the fully qualified name `System.Integer`, you will still get a 16-bit integer.

```
IntegerArray = Array[0..$efffffff] of Integer
```

Generic array of integer.

```
IntPtr = PtrInt
```

A signed integer with the same size in bytes as pointer.

```
jmp_buf = packed record
public
  ebx : LongInt;
  esi : LongInt;
  edi : LongInt;
  bp : Pointer;
  sp : Pointer;
  pc : Pointer;
end
```

Record type to store processor information.

```
Longint = - 2147483648..2147483647
```

A signed 32-bits integer.

```
Longword = 0..4294967295
```

The base 32-bit unsigned type. See the reference manual for more details

`MAKEINTRESOURCE = PChar`

Alias for the PChar (1401) type.

`MarshaledAString = PAnsiChar`

Alias for `PAnsiChar`.

`MarshaledString = PWideChar`

Alias for `PWideChar`.

`NativeInt = PtrInt`

`NativeInt` is defined for Delphi compatibility. It is a signed integer with the size of a pointer, so 32-bit on 32-bit platforms, 64-bit on 64-bit platforms. It is advisable to use unsigned variants of this type. See PtrUInt (1408) for details.

`NativeUInt = PtrUInt`

`NativeUInt` is defined for Delphi compatibility. It is an unsigned integer with the size of a pointer, so 32-bit on 32-bit platforms, 64-bit on 64-bit platforms.

`OpaquePointer = POpaqueData`

Type alias for `POpaqueData`.

`PAnsiChar = PChar`

Alias for PChar (1401) type.

`PAnsiString = ^AnsiString`

Pointer to an ansistring type.

`PBoolean = ^Boolean`

Pointer to a Boolean type.

`PBoolean16 = ^Boolean16`

Pointer to `Boolean16` type.

`PBoolean32 = ^Boolean32`

Pointer to `Boolean32` type.

`PBoolean64 = ^Boolean64`

Pointer to `Boolean64` type.

```
PBoolean8 = ^Boolean8
```

Pointer to `Boolean8` type.

```
PByte = ^Byte
```

Pointer to byte (1396) type.

```
PByteBool = ^ByteBool
```

Pointer to `ByteBool` type.

```
pcalldesc = ^tcalldesc
```

Pointer to TCallDesc (1412) record.

```
PCardinal = ^Cardinal
```

Pointer to Cardinal (1397) type.

```
PChar = ^Char
```

Or the same as a pointer to an array of char. See the reference manual for more information about this type.

```
PClass = ^TClass
```

Pointer to TClass (1412).

```
PCodePointer = ^CodePointer
```

`PCodePointer` is a typed pointer to CodePointer (1397).

```
PCurrency = ^Currency
```

Pointer to currency type.

```
PDate = ^TDateTime
```

Pointer to a TDateTime (1413) type.

```
PDateTime = ^TDateTime
```

Pointer to `TDatetime`.

```
PDispatch = ^IDispatch
```

Pointer to IDispatch (1615) interface type.

```
pdispdesc = ^tdispdesc
```

Pointer to tdispdesc (1413) record.

```
PDouble = ^Double
```

Pointer to double-sized float value.

```
PDWord = ^DWord
```

Pointer to DWord (1397) type.

```
pdynarrayindex = ^tdynarrayindex
```

Pointer to tdynarrayindex (1414) type.

```
pdynarraytypeinfo = ^tdynarraytypeinfo
```

Pointer to TDynArrayTypeInfo (1414) type.

```
PError = ^TError
```

Pointer to an Error (1487) type.

```
PEventState = pointer
```

Pointer to EventState, which is an opaque type.

```
PExceptAddr = ^TExceptAddr
```

`PExceptAddr` is a pointer to PExceptAddr (1402)

```
PExceptObject = ^TExceptObject
```

Pointer to Exception handler procedural type TExceptProc (1416).

```
PExtended = ^Extended
```

Pointer to extended-sized float value.

```
PFileTextRecChar = ^TFileTextRecChar
```

`PFileTextRecChar` is a typed pointer to TFileTextRecChar (1417).

```
PGuid = ^TGuid
```

Pointer to TGUID (1420) type.

```
PInt16 = PSmallInt
```

Pointer to Int16 (1399) type.

```
PInt32 = PLongint
```

Pointer to Int32 (1399) type.

```
PInt64 = ^Int64
```

Pointer to `Int64` type.

```
PInt8 = PShortInt
```

Pointer to Int8 (1399) type.

```
PInteger = ^Integer
```

Pointer to integer (1399) type.

```
PIntegerArray = ^IntegerArray
```

Pointer to IntegerArray (1399) type.

```
PInterface = PUnknown
```

Pointer to IInterface (1399) interface.

```
pinterfaceentry = ^tinterfaceentry
```

Pointer to tinterfaceentry (1610) record.

```
pinterfacetable = ^tinterfacetable
```

Pointer to tinterfacetable (1421) record.

```
PIntPtr = PPtrInt
```

Pointer to IntPtr (1399) type.

```
PJmp_buf = ^jmp_buf
```

Pointer to jmp_buf (1399) record.

```
PLongBool = ^LongBool
```

Pointer to a `LongBool` type.

```
PLongint = ^LongInt
```

Pointer to Longint (1399) type.

```
PLongWord = ^LongWord
```

Pointer to `LongWord` type.

```
PMarshaledAString = ^PAnsiChar
```

Pointer to `PWideChar`.

```
PMarshaledString = ^PWideChar
```

Pointer to `PWideChar`.

```
PMemoryManager = ^TMemoryManager
```

Pointer to TMemoryManager (1422) record.

```
PMethod = ^TMethod
```

Pointer to method.

```
PMsgStrTable = ^TMsgStrTable
```

Pointer to array of TMsgStrTable (1422) records.

```
PNativeInt = ^NativeInt
```

Pointer to NativeInt (1400) type.

```
PNativeUInt = ^NativeUInt
```

Pointer to NativeInt (1400) type.

```
PointerArray = Array[0..512*1024*1024-2] of Pointer
```

Generic pointer array.

```
POleVariant = ^OleVariant
```

Pointer to `OleVariant` type.

```
POpaqueData = ^TOpaqueData
```

`POpaqueData` represents a pointer to data for which the internal structure must not be known. You can make type aliases from this pointer.

```
PPAnsiChar = PPChar
```

Alias for PPChar (1405) type.

```
PPByte = ^PByte
```

`PPByte` is a pointer to a PByte (1401) type.

```
PPChar = ^PChar
```

Pointer to an array of pointers to null-terminated strings.

```
PPCharArray = ^TPCharArray
```

Pointer to TPCharArray (1422) type.

```
PPCodePointer = ^PCodePointer
```

`PPCodePointer` is a typed pointer to PCodePointer (1401).

```
PPDispatch = ^PDispatch
```

Pointer to PDispatch (1401) pointer type.

```
PPDouble = ^PDouble
```

`PPDouble` is a pointer to a PDouble (1402) type.

```
ppdynarraytypeinfo = ^pdynarraytypeinfo
```

Pointer to `pdynarraytypeinfo`.

```
PPLongint = ^PLongint
```

`PPLongint` is a pointer to a PLongint (1403) type.

```
PPointer = ^Pointer
```

Pointer to a pointer type.

```
PPointerArray = ^PointerArray
```

Pointer to PointerArray (1404) type.

```
PPPAnsiChar = PPPChar
```

`PPPAnsiChar` is a typed pointer to PPAnsichar (1404).

```
PPPChar = ^PPChar
```

`PPPChar` is a pointer to a PPChar (1405)

```
PPPointer = ^PPointer
```

Pointer to a PPointer (1405) type.

```
PPPWideChar = ^PPWideChar
```

`PPPWideChar` is a pointer to a PPWideChar (1406) type.

```
PPtrInt = ^PtrInt
```

Pointer to PtrInt (1407) type.

```
PPtrUInt = ^PtrUInt
```

Pointer to unsigned integer of pointer size.

```
PPUnknown = ^PUnknown
```

Pointer to untyped pointer.

```
PPVmt = ^PVmt
```

Pointer to `PVMT` pointer.

```
PPWideChar = ^PWideChar
```

Pointer to link id="PWideChar"> type.

```
PQWord = ^QWord
```

Pointer to `QWord` type.

```
PQWordBool = ^QWordBool
```

Pointer to `QWordBool` type.

```
PRawByteString = ^RawByteString
```

Pointer to `RawByteString`.

```
PRTLCriticalSection = ^TRTLCriticalSection
```

Pointer to #rtl.system.TRTLCriticalSection (1424) type.

```
PRTLEvent = pointer
```

Pointer to RTLEvent, which is an opaque type.

```
PShortInt = ^ShortInt
```

Pointer to shortint (1410) type.

```
PShortString = ^ShortString
```

Pointer to a shortstring type.

```
PSingle = ^Single
```

Pointer to single-sized float value.

```
PSizeInt = ^SizeInt
```

Pointer to a SizeInt (1410) type.

```
PSizeUInt = ^SizeUInt
```

PSizeUInt64 is a pointer to a SizeUInt (1411), an unsigned integer of architecture-dependent size.

```
PSmallInt = ^SmallInt
```

Pointer to smallint (1411) type.

```
pstringmessagetable = ^TStringMessageTable
```

Pointer to TStringMessageTable (1426) record.

```
PText = ^Text
```

Pointer to text file.

```
PtrInt = Int64
```

Ptrint is a signed integer type which has always the same size as a pointer. Ptrint is considered harmful and should almost never be used in actual code, because pointers are normally unsigned. For example, consider the following code:

```
getmem(p,2048);                 {Assume the address of p becomes $7ffffff0.}
q:=pointer(ptrint(p)+1024));    {Overflow error.}
writeln(q>p);                   {Incorrect answer.}
```

Ptrint might have a valid use when two pointers are subtracted from each other if it is unknown which pointer has the largest address. However, even in this case ptrint causes trouble in case the distance is larger than high(ptrint) and must be used with great care.

The introduction of the ptrint type was a mistake. Please use PtrUInt (1408) instead.

```
PtrInt = LongInt
```

Ptrint is a signed integer type which has always the same size as a pointer. Ptrint is considered harmful and should almost never be used in actual code, because pointers are normally unsigned. For example, consider the following code:

```
getmem(p,2048);                 {Assume the address of p becomes $7ffffff0.}
q:=pointer(ptrint(p)+1024));    {Overflow error.}
writeln(q>p);                   {Incorrect answer.}
```

Ptrint might have a valid use when two pointers are subtracted from each other if it is unknown which pointer has the largest address. However, even in this case ptrint causes trouble in case the distance is larger than high(ptrint) and must be used with great care.

The introduction of the ptrint type was a mistake. Please use PtrUInt (1408) instead.

```
PtrUInt = DWord
```

PtrUInt is an unsigned integer type which has always the same size as a pointer. When using integers which will be cast to pointers and vice versa, use this type, never the regular Cardinal type.

```
PtrUInt = QWord
```

PtrUInt is an unsigned integer type which has always the same size as a pointer. When using integers which will be cast to pointers and vice versa, use this type, never the regular Cardinal type.

```
PUCS2Char = PWideChar
```

Pointer to UCS2Char (1440) character.

```
PUCS4Char = ^UCS4Char
```

Pointer to UCS4Char (1440).

```
PUCS4CharArray = ^TUCS4CharArray
```

Pointer to array of UCS4Char (1440) characters.

```
PUInt16 = PWord
```

Pointer to UInt16 (1440) type.

```
PUInt32 = PDWord
```

Pointer to UInt32 (1440) type.

```
PUInt64 = ^UInt64
```

PUInt64 is a pointer to a UInt64 (1440), an unsigned 64-bit integer.

```
PUInt8 = PByte
```

Pointer to UInt8 (1440) type.

```
PUintPtr = PPtrUInt
```

Pointer to UIntPtr (1440) type.

```
PUnicodeChar = ^UnicodeChar
```

PUnicodeChar is a pointer to a Unicode character, just like PChar is a pointer to a Char ansistring character.

```
PUnicodeString = ^UnicodeString
```

PUnicodeString is a pointer to a UnicodeString string.

```
PUnknown = ^IUnknown
```

Untyped pointer.

```
PUTF8Char = PAnsiChar
```

Pointer to `UTF8Char`.

```
PUTF8String = ^UTF8String
```

Pointer to UTF8String (1441).

```
pvararray = ^tvararray
```

Pointer to TVarArray (1432) type.

```
pvararraybound = ^tvararraybound
```

Pointer to tvararraybound (1432) type.

```
pvararrayboundarray = ^tvararrayboundarray
```

Pointer to tvararrayboundarray (1432) type.

```
pvararraycoorarray = ^tvararraycoorarray
```

Pointer to tvararraycoorarray (1432) type.

```
pvardata = ^tvardata
```

Pointer to TVarData (1435) record.

```
PVariant = ^Variant
```

Pointer to `Variant` type.

```
pvariantmanager = ^tvariantmanager
```

Pointer to TVariantManager (1437) record.

```
PVarRec = ^TVarRec
```

Pointer to TVarRec (1439) type.

```
PVmt = ^TVmt
```

Pointer to TVMT (1615) record.

```
PWideChar = ^WideChar
```

Pointer to WChar (1441).

```
PWideString = ^WideString
```

Pointer to widestring type.

```
PWord = ^Word
```

Pointer to word (1441) type.

```
PWordBool = ^WordBool
```

Pointer to a `WordBool` type.

```
QWord = 0..18446744073709551615
```

`QWord` is a 64-bit, unsigned integer type, with range [0..18446744073709551615].

```
RawByteString = ansistring
```

`RawByteString` is a single-byte character string which does not have any codepage associated with it.

This means that assigning a single-byte character string to this kind of string will not change the codepage of the string.

Inversely, when assigning a `RawByteString` to a single-byte string, the codepage of the destination is simply set to the codepage of the rawbytestring: no codepage conversion happens, the reference count is simply increased.

```
Real = Double
```

Alias for real type.

```
Real48 = Array[0..5] of Byte
```

TP compatible real type (6 bytes) definition.

```
Shortint = - 128..127
```

A signed 8-bits integer.

```
SizeInt = Int64
```

`SizeInt` is used to describe sizes of structures in FPC using a signed integer. It is an opaque type: The actual type of this type depends on the architecture: its size reflects the maximum addressable memory on the current architecture, thus it is 64-bit on 64-bit platforms, 32-bit on 32-bit platforms, and 16 bit on 16 bit platforms.

If you wish to create cross-platform code, do not use the actual type of `SizeInt` displayed here, i t reflects the definition used on the platform used to generate the documentation.

```
SizeInt = LongInt
```

`SizeInt` is used to describe sizes of structures in FPC using a signed integer. It is an opaque type: The actual type of this type depends on the architecture: its size reflects the maximum addressable memory on the current architecture, thus it is 64-bit on 64-bit platforms, 32-bit on 32-bit platforms, and 16 bit on 16 bit platforms.

If you wish to create cross-platform code, do not use the actual type of `SizeInt` displayed here, i t reflects the definition used on the platform used to generate the documentation.

```
SizeUInt = QWord
```

SizeUInt is used to describe sizes of structures in FPC using an unsigned integer. The actual type of this type depends on the architecture: its size reflects the maximum addressable memory on the current architecture, thus it is 64-bit on 64-bit platforms, 32-bit on 32-bit platforms, and 16 bit on 16 bit platforms.

```
SizeUInt = DWord
```

SizeUInt is used to describe sizes of structures in FPC using an unsigned integer. The actual type of this type depends on the architecture: its size reflects the maximum addressable memory on the current architecture, thus it is 64-bit on 64-bit platforms, 32-bit on 32-bit platforms, and 16 bit on 16 bit platforms.

```
Smallint = - 32768..32767
```

A signed 16-bits integer.

```
TAbstractErrorProc = procedure
```

Abstract error handler procedural type.

```
TAllocateThreadVarsHandler = procedure
```

Threadvar allocation callback type for TThreadManager (1428).

```
TAnsiChar = Char
```

Alias for 1-byte sized char.

```
TAssertErrorProc = procedure(const msg: ShortString;
  const fname: ShortString; lineno: LongInt;
  erroraddr: pointer)
```

Assert error handler procedural type.

```
TBacktraceStrFunc = function(Addr: CodePointer) : ShortString
```

Type for formatting of backtrace dump.

```
TBasicEventCreateHandler = function(EventAttributes: Pointer;
  AManualReset: Boolean;
  InitialState: Boolean;
  const Name: ansistring)
  : PEventState
```

callback type for creating eventstate in TThreadManager (1428).

```
TBasicEventHandler = procedure(state: PEventState)
```

Generic callback type for handling eventstate in TThreadManager (1428).

```
TBasicEventWaitForHandler = function(timeout: Cardinal;
  state: PEventState) : LongInt
```

Wait for basic event callback type for TThreadManager (1428).

```
TBeginThreadHandler = function(sa: Pointer; stacksize: PtrUInt;
  ThreadFunction: TThreadFunc; p: pointer
  ;
                                creationFlags: DWord;
  var ThreadId: TThreadID) : TThreadID
```

Callback for thread start in TThreadManager (1428).

```
TBoundArray = Array of SizeInt
```

Dynamic array of integer.

```
tcalldesc = packed record
public
  calltype : Byte;
  argcount : Byte
  ;
  namedargcount : Byte;
  argtypes : Array[0..255] of Byte;
end
```

`tcalldesc` is used to encode the arguments to a dispatch call to an OLE dual interface. It is used on windows only. It describes the arguments to a call.

```
TClass = Class of TObject
```

Class of TObject (1623).

```
TCompareOption = (coIgnoreCase)
```

Table 75.10: Enumeration values for type TCompareOption

Value	Explanation
coIgnoreCase	Ignore case (usually identical to coLingIgnoreCase).

`TCompareOption` indicates how 2 strings should be compared. This option is used in the WideStringManager (1440) implementation when comparing 2 strings. The following options exist:

Ignore case linguistically (usually identical to coIgnoreCase).

Ignore diacritic characters.

Ignore case (usually identical to coLingIgnoreCase).

Corresponding hiragana and katakana characters compare as equal.

Corresponding hiragana and katakana characters compare as equal.

Ignore nonspace characters (usually identical to coLingIgnoreDiacritic).

Ignore symbols and punctuation characters.

Ignore half-width and full width characters (used in Chinese and Japanese).

Use linguistic rules for casing, instead of file system rules.

Treat digits as numbers (20 before 120).

Handle punctuation as symbols.

Not all platforms will support all possibilities.

```
TCompareOptions = Set of TCompareOption
```

`TCompareOptions` is simply a set of TCompareOption (1412) enumeration values.

```
TCriticalSectionHandler = procedure(var cs)
```

Generic callback type for critical section handling in TThreadManager (1428).

```
TCriticalSectionHandlerTryEnter = function(var cs) : LongInt
```

`TCriticalSectionHandlerTryEnter` is the function prototype for the TryEnterCriticalsection (1591) function, in the TThreadManager (1428) record's `TryEnterCriticalSection` field.

```
TCtrlBreakHandler = function(CtrlBreak: Boolean) : Boolean
```

`TCtrlBreakHandler` is the prototype for the CTRL-C handler. If `CtrlBreak` is `True` then Ctrl-Break was hit, otherwise `CTRL-C` was hit. The handlers should return `True` to signal that the key-combination was handled. If `False` is returned, then default handling will be used, which by default means an exception will be raised if the sysutils unit is used.

```
TDate = TDateTime
```

`TDate` is defined for Delphi compatibility. This type is deprecated, use TDateTime (1413) instead.

```
TDateTime = Double
```

Encoded Date-Time type.

```
tdispdesc = packed record
public
  dispid : LongInt;
  restype : Byte
  ;
  calldesc : tcalldesc;
end
```

`tcalldesc` is used to encode a dispatch call to an OLE dispatch interface. It is used on windows only. It describes the dispatch call call.

```
tdynarrayindex = SizeInt
```

A variable of type tdynarrayindex will always have the correct size, suitable for serving as an index in a dynamic array.

```
tdynarraytypeinfo = packed record
public
  kind : TTypeKind;
  namelen
  : Byte;
  elesize : SizeInt;
  eletype : ppdynarraytypeinfo;
  vartype
  : LongInt;
end
```

tdynarraytypeinfo describes the structure of a multi-dimensional dynamic array. It is used in the DynArraySetLength (1482) call.

```
TDynLibsManager = record
public
  LoadLibraryU : TLoadLibraryUHandler
  ;
  LoadLibraryA : TLoadLibraryAHandler;
  GetProcAddress : TGetProcAddressHandler
  ;
  GetProcAddressOrdinal : TGetProcAddressOrdinalHandler;
  UnloadLibrary
  : TUnloadLibraryHandler;
  GetLoadErrorStr : TGetLoadErrorStrHandler
  ;
end
```

TDynLibsManager contains all the callbacks needed to load and manage dynamic libraries. The system unit does not contain dynamic loading library support on all supported platforms. Like the unicode string support, heap support and thread support, support for loading dnamic libraries is pluggable. This record contains the necessary callbacks that the system unit needs to implement loading of dynamic libraries (needed for example for run-time package support).

Including the dynlibs (734) unit will enable support for dynamically loadable libraries on all platforms that support this.

```
TEndThreadHandler = procedure(ExitCode: DWord)
```

Callback for thread end in TThreadManager (1428).

```
TEntryInformation = record
public
  InitFinalTable : Pointer;
  ThreadvarTablesTable
  : Pointer;
  ResourceStringTables : Pointer;
  ResStrInitTables
```

```
  : Pointer;
  ResLocation : Pointer;
  PascalMain : procedure;
  valgrind_used
  : Boolean;
  OS : TEntryInformationOS;
end
```

`TEntryInformation` is used to initialize a Free Pascal program or library. Under normal circumstances, there should be no need to use this structure directly: it is used by the system unit and special linking units.

```
TEntryInformationOS = record
public
  argc : LongInt;
  argv : PPChar
  ;
  envp : PPChar;
  stkptr : pointer;
  stklen : SizeUInt;
  haltproc
  : procedure(e: LongInt);
end
```

`TEntryInformationOS` represents executable entry information for the current OS. This structure is OS dependent.

```
TError = LongInt
```

Error type, used in variants.

```
TErrorProc = procedure(ErrNo: LongInt; Address: CodePointer;
  Frame: Pointer)
```

Standard error handler procedural type.

```
TExceptAddr = record
public
  buf : PJmp_buf;
  next : PExceptAddr
  ;
  frametype : LongInt;
end
```

`TExceptAddr` describes an exception frame on the exception address frame stack. It is used in structured stack exception handling.

```
TExceptObject = record
public
  FObject : TObject;
  Addr : CodePointer
  ;
```

```
  Next : PExceptObject;
  refcount : LongInt;
  Framecount : LongInt
  ;
  Frames : PCodePointer;
end
```

`TExceptObject` is the exception description record which is found on the exception stack.

```
TExceptProc = procedure(Obj: TObject; Addr: CodePointer;
  FrameCount: LongInt; Frame: PCodePointer)
```

Exception handler procedural type.

```
TextBuf = Array[0..TextRecBufSize-1] of AnsiChar
```

`TextBuf` is a type for the default buffer used in TextRec (1416).

```
TextFile = Text
```

Alias for `Text` file type.

```
TextRec = record
public
  Handle : THandle;
  Mode : LongInt;
  bufsize
  : SizeInt;
  _private : SizeInt;
  bufpos : SizeInt;
  bufend : SizeInt
  ;
  bufptr : ^TextBuf;
  openfunc : CodePointer;
  inoutfunc : CodePointer
  ;
  flushfunc : CodePointer;
  closefunc : CodePointer;
  UserData
  : Array[1..32] of Byte;
  name : Array[0..textrecnamelength-1] of
  TFileTextRecChar;
  LineEnd : TLineEndStr;
  buffer : TextBuf;
end
```

`TextRec` is the underlying type used in text files. It should be treated as opaque and never manipulated directly.

```
TFileTextRecChar = UnicodeChar
```

TFileTextRecChar is the type of character used in TextRec (1416) or FileRec (1398) file types. It is an alias type, depending on platform and RTL compilation flags. No assumptions should be made on the actual character type.

```
TFloatSpecial = (fsZero,fsNZero,fsDenormal,fsNDenormal,fsPositive
  ,
                 fsNegative,fsInf,fsNInf,fsNaN,fsInvalidOp)
```

Table 75.11: Enumeration values for type TFloatSpecial

Value	Explanation
fsDenormal	Denormal value.
fsInf	Infinity.
fsInvalidOp	Invalid operation.
fsNaN	Not a number.
fsNDenormal	Negative enormal value.
fsNegative	Negative value.
fsNInf	Negative infinity.
fsNZero	Negative zero.
fsPositive	Positive value.
fsZero	Zero.

TFloatSpecial enumerates a series of floating point value properties.

```
TFPCHeapStatus = record
public
  MaxHeapSize : PtrUInt;
  MaxHeapUsed
  : PtrUInt;
  CurrHeapSize : PtrUInt;
  CurrHeapUsed : PtrUInt;
  CurrHeapFree : PtrUInt;
end
```

TFPCHeapStatus describes the state of the FPC heap manager. This is not equivalent to the THeapStatus (1420) record defined by Delphi, which contains information not meaningful for the FPC heap manager. The heap status can be retrieved by the GetFPCHeapStatus (1502) call.

```
TFPResourceHandle = PtrUInt
```

TFPResourceHandle represents a handle to a binary resource and is used in the various resource calls. Its actual type and size may differ across platforms.

```
TFPResourceHGLOBAL = PtrUInt
```

TFPResourceHGLOBAL represents a handle to the global module containing a resource. It is used in the various resource calls. It is an opaque type: its actual type and size may differ across platforms.

```
TFPResourceHMODULE = PtrUInt
```

TFPResourceHMODULE represents a module (library, executable, other) in which a resource is located. It is used in the various resource calls. It is an opaque type: its actual type and size may differ across platforms.

```
TFPUException = (exInvalidOp,exDenormalized,exZeroDivide,exOverflow
  ,
                 exUnderflow,exPrecision)
```

Table 75.12: Enumeration values for type TFPUException

Value	Explanation
exDenormalized	
exInvalidOp	Invalid operation error.
exOverflow	Float overflow error.
exPrecision	Precision error.
exUnderflow	Float underflow error.
exZeroDivide	Division by zero error.

TFPUException describes what floating point errors raise exceptions. It has been moved here from the Math unit.

```
TFPUExceptionMask = Set of TFPUException
```

TFPUExceptionMask is a set of TFPUException constants

```
TFPUPrecisionMode = (pmSingle,pmReserved,pmDouble,pmExtended)
```

Table 75.13: Enumeration values for type TFPUPrecisionMode

Value	Explanation
pmDouble	Double-type precision.
pmExtended	Extended-type precision.
pmReserved	?
pmSingle	Single-type precision.

TFPUPrecisionMode describes the possible default precisions for the software Floating Point math routines. It has been moved here from the math unit.

```
TFPURoundingMode = (rmNearest,rmDown,rmUp,rmTruncate)
```

Table 75.14: Enumeration values for type TFPURoundingMode

Value	Explanation
rmDown	Round to biggest integer smaller than value.
rmNearest	Round to nearest integer.
rmTruncate	Cut off fractional part.
rmUp	Round to smallest integer larger than value.

TFPURoundingMode enumerates the possible values for software floating point math rounding. It has been moved here from the math unit.

```
TGetCurrentThreadIdHandler = function : TThreadID
```

Callback type for retrieving thread ID in TThreadManager (1428).

```
TGetLoadErrorStrHandler = function : string
```

TGetLoadErrorStrHandler is the type for the GetLoadErrorStr call using ansistring names in the TDynLibsManager (1414) dynamic library loading manager.

```
TGetProcAddressHandler = function(Lib: TLibHandle;
  const ProcName: AnsiString) : Pointer
```

TGetProcAddressHandler is the type for the GetProcAddress call using ansistring names in the TDynLibsManager (1414) dynamic library loading manager.

```
TGetProcAddressOrdinalHandler = function(Lib: TLibHandle;
  Ordinal: TOrdinalEntry)
  : Pointer
```

TGetProcAddressOrdinalHandler is the type for the GetProcAddressOrdinal call using ansistring names in the TDynLibsManager (1414) dynamic library loading manager.

```
TGuid = packed record
case Integer of
1: (
public
  Data1 : DWord
  ;
  Data2 : Word;
  Data3 : Word;
  Data4 : Array[0..7] of Byte;
  );
2: (
public
  D1 : DWord;
  D2 : Word;
  D3 : Word;
  D4 : Array
  [0..7] of Byte;
);
3: (
public
  time_low : DWord;
  time_mid : Word
  ;
  time_hi_and_version : Word;
  clock_seq_hi_and_reserved : Byte
  ;
  clock_seq_low : Byte;
  node : Array[0..5] of Byte;
);
end
```

Standard GUID representation type.

```
THandle = LongInt
```

This type should be considered opaque. It is used to describe file and other handles.

```
THeapStatus = record
public
  TotalAddrSpace : Cardinal;
  TotalUncommitted
  : Cardinal;
  TotalCommitted : Cardinal;
  TotalAllocated : Cardinal
  ;
  TotalFree : Cardinal;
  FreeSmall : Cardinal;
  FreeBig : Cardinal
  ;
  Unused : Cardinal;
  Overhead : Cardinal;
  HeapErrorCode : Cardinal
  ;
end
```

`THeapStatus` is the record describing the current heap status. It is returned by the GetHeapStatus (1502) call.

```
TInitThreadVarHandler = procedure(var offset: DWord; size: DWord)
```

Threadvar initialization callback type for TThreadManager (1428).

```
TInterfacedClass = Class of TInterfacedObject
```

`TInterfacedClass` is a descendent of

```
tinterfaceentrytype = (etStandard,etVirtualMethodResult,
  etStaticMethodResult,etFieldValue,
  etVirtualMethodClass,etStaticMethodClass,
  etFieldValueClass)
```

Table 75.15: Enumeration values for type tinterfaceentrytype

Value	Explanation
etFieldValue	Field value.
etFieldValueClass	Interface provided by a class field.
etStandard	Standard entry.
etStaticMethodClass	Interface provided by a static class method.
etStaticMethodResult	Static method.
etVirtualMethodClass	Interface provided by a virtual class method.
etVirtualMethodResult	Virtual method.

This is an internal type for the compiler to encode calls to dispatch interfaces.

```
tinterfacetable = record
public
  EntryCount : SizeUInt;
  Entries
  : Array[0..0] of tinterfaceentry;
end
```

Record to store list of interfaces of a class.

```
TLibHandle = PtrInt
```

TLibHandle should be considered an opaque type. It is defined differently on various platforms. The definition shown here depends on the platform for which the documentation was generated.

```
TLineEndStr = string
```

TLineEndStr is an alias for the actual line ending string type, used in TextRec (1416). It should be treated as opaque.

```
TLoadLibraryAHandler = function(const Name: RawByteString) :
  TLibHandle
```

TLoadLibraryAHandler is the type for the loadlibrary call using ansistring names in the TDynLibsManager (1414) dynamic library loading manager.

```
TLoadLibraryUHandler = function(const Name: UnicodeString) :
  TLibHandle
```

TLoadLibraryUHandler is the type for the loadlibrary call using unicode names in the TDynLibsManager (1414) dynamic library loading manager.

```
TMemoryManager = record
public
  NeedLock : Boolean;
  Getmem : function
  (Size: PtrUInt) : Pointer;
  Freemem : function(p: pointer) : PtrUInt
  ;
  FreememSize : function(p: pointer; Size: PtrUInt) : PtrUInt;
  AllocMem : function(Size: PtrUInt) : Pointer;
  ReAllocMem : function
  (var p: pointer; Size: PtrUInt) : Pointer;
  MemSize : function(p
  : pointer) : PtrUInt;
  InitThread : procedure;
  DoneThread : procedure
  ;
  RelocateHeap : procedure;
  GetHeapStatus : function : THeapStatus
  ;
  GetFPCHeapStatus : function : TFPCHeapStatus;
end
```

TMemoryManager describes the memory manager. For more information about the memory manager, see the programmer's reference.

```
TMethod = record
public
  Code : CodePointer;
  Data : Pointer;
end
```

TMethod describes a general method pointer, and is used in Run-Time Type Information handling.

```
TMsgStrTable = record
public
  name : PShortString;
  method : CodePointer
  ;
end
```

Record used in string message handler table.

```
TOpaqueData = record
end
```

TOpaqueData represents data for which the internal structure must not be known. It is mustly useful for the pointer definition POpaqueData (1404).

```
TOrdinalEntry = SizeUInt
```

Ordinal of entry point (windows only).

```
TOSTimestamp = Int64
```

```
packed TPCharArray = Array[0..(MaxLongintdivSizeOf(PChar))-1] of
  PChar
```

Array of PChar.

```
TProcedure = procedure
```

Simple procedural type.

```
TReleaseThreadVarsHandler = procedure
```

Threadvar release callback type for TThreadManager (1428).

```
TRelocateThreadVarHandler = function(offset: DWord) : pointer
```

Threadvar relocalization callback type for TThreadManager (1428).

```
TResourceHandle = PtrUInt
```

This is an opaque type.

```
TResourceManager = record
public
  HINSTANCEFunc : function : TFPResourceHMODULE
  ;
  EnumResourceTypesFunc : function(ModuleHandle: TFPResourceHMODULE
  ; EnumFunc: EnumResTypeProc;
           lParam: PtrInt) : LongBool
  ;
  EnumResourceNamesFunc : function(ModuleHandle: TFPResourceHMODULE
  ; ResourceType: PChar;
           EnumFunc: EnumResNameProc; lParam
  : PtrInt) : LongBool;
  EnumResourceLanguagesFunc : function(ModuleHandle
  : TFPResourceHMODULE; ResourceType: PChar;
           ResourceName
  : PChar; EnumFunc: EnumResLangProc; lParam: PtrInt)
            :
  LongBool;
  FindResourceFunc : function(ModuleHandle: TFPResourceHMODULE
  ; ResourceName: PChar;
           ResourceType: PChar) : TFPResourceHandle
  ;
  FindResourceExFunc : function(ModuleHandle: TFPResourceHMODULE
  ; ResourceType: PChar;
           ResourceName: PChar; Language:
  Word) : TFPResourceHandle;
  LoadResourceFunc : function(ModuleHandle
  : TFPResourceHMODULE; ResHandle: TFPResourceHandle)
            :
  TFPResourceHGLOBAL;
  SizeofResourceFunc : function(ModuleHandle
  : TFPResourceHMODULE; ResHandle: TFPResourceHandle)
            :
  LongWord;
  LockResourceFunc : function(ResData: TFPResourceHGLOBAL
  ) : Pointer;
  UnlockResourceFunc : function(ResData: TFPResourceHGLOBAL
  ) : LongBool;
  FreeResourceFunc : function(ResData: TFPResourceHGLOBAL
  ) : LongBool;
end
```

`TResourceManager` is the record describing the resource manager. Depending on the kind of resources (internal, external), another resource managing handler is installed by the system. The resource manager record is used by all resource handling functions to do the actual work: for each function in the API, a handler function is available. People wishing to implement their own resource manager, must implement all handler functions in their implementation.

As soon as resources are used, the compiler will install a resource manager, depending on the platform, this may be an internal or an external resource manager.

```
TRTLCreateEventHandler = function : PRTLEvent
```

Callback type for creating a `TRTLEvent` type in TThreadManager (1428).

```
TRTLCRITICALSECTION = Opaque type
```

`TRTLCriticalSection` represents a critical section (a mutex). This is an opaque type, it can differ from operating system to operating system. No assumptions should be made about it's structure or contents.

```
TRTLEventHandler = procedure(AEvent: PRTLEvent)
```

Generic TRTLEvent handling type for TThreadManager (1428).

```
TRTLEventHandlerTimeout = procedure(AEvent: PRTLEvent; timeout: LongInt
  )
```

TRTLEvent timeout handling type for TThreadManager (1428).

```
trtlmethod = procedure of object
```

Callback type for synchronization event.

```
TRuntimeError = (reNone,reOutOfMemory,reInvalidPtr,reDivByZero,
  reRangeError,reIntOverflow,reInvalidOp,reZeroDivide
  ,
                reOverflow,reUnderflow,reInvalidCast,reAccessViolation
  ,
                rePrivInstruction,reControlBreak,reStackOverflow
  ,
                reVarTypeCast,reVarInvalidOp,reVarDispatch,
  reVarArrayCreate,reVarNotArray,reVarArrayBounds,
  reAssertionFailed,reExternalException,reIntfCastError
  ,
                reSafeCallError,reQuit,reCodesetConversion,
  reNoDynLibsSupport,reThreadError)
```

Table 75.16: Enumeration values for type TRuntimeError

Value	Explanation
reAccessViolation	Access Violation.
reAssertionFailed	Assertion failed error.
reCodesetConversion	Code set conversion error.
reControlBreak	User pressed CTRL-C.
reDivByZero	Division by zero error.
reExternalException	An external exception occurred.
reIntfCastError	Interface typecast error.
reIntOverflow	Integer overflow error.
reInvalidCast	Invalid (class) typecast error.
reInvalidOp	Invalid operation error.
reInvalidPtr	Invalid pointer error.
reNoDynLibsSupport	Runtime error if no dynamic library support is available.
reNone	No error.
reOutOfMemory	Out of memory error.
reOverflow	Overflow error.
rePrivInstruction	Privileged instruction error.
reQuit	Quit signal error.
reRangeError	Range check error.
reSafeCallError	Safecall (IDispInterface) error.
reStackOverflow	Stack overflow error.
reThreadError	Runtime error if no thread support is available.
reUnderflow	Underflow error.
reVarArrayBounds	Variant array bounds error.
reVarArrayCreate	Variant array creation error.
reVarDispatch	Variant Dispatch error.
reVarInvalidOp	Invalid variant operation error.
reVarNotArray	Variant is not an array error.
reVarTypeCast	Invalid typecase from variant.
reZeroDivide	Division by zero error.

`TRuntimeError` is used in the Error (1487) procedure to indicate what kind of error should be reported.

```
TSafeCallErrorProc = procedure(error: HRESULT; addr: pointer)
```

Prototype of a safecall error handler routine. `Error` is the error number (passed by the Windows operating system) and `Addr` is the address where the error occurred.

```
TSemaphoreDestroyHandler = procedure(const sem: Pointer)
```

`TSemaphoreDestroyHandler` is the function prototype to destroy an existing semaphore, as returned by (`ThreadManager.SemaphoreInit`). It is used by the thread manager (`ThreadManager.SemaphoreDest`

```
TSemaphorePostHandler = procedure(const sem: Pointer)
```

`TSemaphorePostHandler` is the function prototype to post an event to the semaphore. It should handle a pointer as returned by the `ThreadManager.SemaphoreInit` procedure. it's used by the thread manager `ThreadManager.SemaphorePost`.

```
TSemaphoreWaitHandler = procedure(const sem: Pointer)
```

TSemaphoreWaitHandler is the function prototype to wait on an event on the semaphore (which should be posted to the semaphore with ThreadManager.SemaphorePost). It should handle a pointer as returned by the ThreadManager.SemaphoreInit procedure. it's used by the thread manager ThreadManager.SemaphoreWait.

```
TSempahoreInitHandler = function : Pointer
```

TSempahoreInitHandler is the function prototype for initializing a semaphore. It is used by the thread manager (ThreadManager.SemaphoreInit) to create semaphores. The function should return a pointer, usable by the other semaphore functions of the thread manager.

```
TStandardCodePageEnum = (scpAnsi,scpConsoleInput,scpConsoleOutput
  ,
                         scpFileSystemSingleByte)
```

Table 75.17: Enumeration values for type TStandardCodePageEnum

Value	Explanation
scpAnsi	Ansi codepage (CP_ACP).
scpConsoleInput	Console input codepage.
scpConsoleOutput	Console output codepage.
scpFileSystemSingleByte	File system single byte codepage.

TStandardCodePageEnum describes several types of standard used codepages, which can be queried by the unicode string manager TUnicodeStringManager (1431).

```
TStringMessageTable = record
public
  count : LongInt;
  msgstrtable
  : Array[0..0] of TMsgStrTable;
end
```

Record used to describe the string messages handled by a class. It consists of a count, followed by an array of TMsgStrTable (1422) records.

```
TSystemCodePage = Word
```

TSystemCodePage is a type used to indicate code pages. It should be treated as an opaque type.

```
TTextBuf = TextBuf
```

TTextBuf is an alias for TextBuf

```
TTextLineBreakStyle = (tlbsLF,tlbsCRLF,tlbsCR)
```

Table 75.18: Enumeration values for type TTextLineBreakStyle

Value	Explanation
tlbsCR	Carriage-return (#13, Mac-OS style).
tlbsCRLF	Carriage-return, line-feed (#13#30, Windows style).
tlbsLF	Line-feed only (#10, Unix style).

Text line break style. (end of line character).

```
TThreadFunc = function(parameter: pointer) : PtrInt
```

Thread function prototype.

```
TThreadGetPriorityHandler = function(threadHandle: TThreadID) :
  LongInt
```

Callback type for thread priority getting in TThreadManager (1428).

```
TThreadHandler = function(threadHandle: TThreadID) : DWord
```

Generic thread handler callback for TThreadManager (1428).

```
TThreadID = PtrUInt
```

This is an opaque type, it can differ from operating system to operating system.

```
TThreadManager = record
public
  InitManager : function : Boolean
  ;
  DoneManager : function : Boolean;
  BeginThread : TBeginThreadHandler
  ;
  EndThread : TEndThreadHandler;
  SuspendThread : TThreadHandler
  ;
  ResumeThread : TThreadHandler;
  KillThread : TThreadHandler;
  CloseThread : TThreadHandler;
  ThreadSwitch : TThreadSwitchHandler
  ;
  WaitForThreadTerminate : TWaitForThreadTerminateHandler;
  ThreadSetPriority
  : TThreadSetPriorityHandler;
  ThreadGetPriority : TThreadGetPriorityHandler
  ;
  GetCurrentThreadId : TGetCurrentThreadIdHandler;
  SetThreadDebugNameA
  : TThreadSetThreadDebugNameHandlerA;
  SetThreadDebugNameU : TThreadSetThreadDebugNameHandlerU
  ;
```

```
  InitCriticalSection : TCriticalSectionHandler;
  DoneCriticalSection
  : TCriticalSectionHandler;
  EnterCriticalSection : TCriticalSectionHandler
  ;
  TryEnterCriticalSection : TCriticalSectionHandlerTryEnter;
  LeaveCriticalSection
  : TCriticalSectionHandler;
  InitThreadVar : TInitThreadVarHandler
  ;
  RelocateThreadVar : TRelocateThreadVarHandler;
  AllocateThreadVars
  : TAllocateThreadVarsHandler;
  ReleaseThreadVars : TReleaseThreadVarsHandler
  ;
  BasicEventCreate : TBasicEventCreateHandler;
  BasicEventDestroy
  : TBasicEventHandler;
  BasicEventResetEvent : TBasicEventHandler
  ;
  BasicEventSetEvent : TBasicEventHandler;
  BasiceventWaitFOr
  : TBasicEventWaitForHandler;
  RTLEventCreate : TRTLCreateEventHandler
  ;
  RTLEventDestroy : TRTLEventHandler;
  RTLEventSetEvent : TRTLEventHandler
  ;
  RTLEventResetEvent : TRTLEventHandler;
  RTLEventWaitFor : TRTLEventHandler
  ;
  RTLEventWaitForTimeout : TRTLEventHandlerTimeout;
end
```

`TThreadManager` is a record that contains all callbacks needed for the thread handling routines of the Free Pascal Run-Time Library. The thread manager can be set by the SetThreadManager (1572) procedure, and the current thread manager can be retrieved with the GetThreadManager (1505) procedure.

The Windows RTL will set the thread manager automatically to a system thread manager, based on the Windows threading routines. Unix operating systems provide a unit cthreads which implements threads based on the C library POSIX thread routines. It is not included by default, because it would make the system unit dependent on the C library.

For more information about thread programming, see the programmer's guide.

```
TThreadSetPriorityHandler = function(threadHandle: TThreadID;
  Prio: LongInt) : Boolean
```

Callback type for thread priority setting in TThreadManager (1428).

```
TThreadSetThreadDebugNameHandlerA = procedure(threadHandle: TThreadID
  ;
                                                const ThreadName
  : AnsiString)
```

TThreadSetThreadDebugNameHandlerA is the type of the TThreadManager (1428)SetThreadDebugNameA method. It receives the thread ID in threadHandle and the name (as AnsiString) to apply in ThreadName.

```
TThreadSetThreadDebugNameHandlerU = procedure(threadHandle: TThreadID
  ;
                                                const ThreadName
  : UnicodeString)
```

TThreadSetThreadDebugNameHandlerU is the type of the TThreadManager (1428)SetThreadDebugNameU method. It receives the thread ID in threadHandle and the name to apply (as unicodeString) in ThreadName.

```
TThreadSwitchHandler = procedure
```

Callback type for thread switch in TThreadManager (1428).

```
TTime = TDateTime
```

TTime is defined for Delphi compatibility. This type is deprecated, use TDateTime (1413) instead.

```
TTypeKind = (tkUnknown,tkInteger,tkChar,tkEnumeration,tkFloat,tkSet
  ,
             tkMethod,tkSString,tkLString,tkAString,tkWString,tkVariant
  ,
             tkArray,tkRecord,tkInterface,tkClass,tkObject,tkWChar
  ,
             tkBool,tkInt64,tkQWord,tkDynArray,tkInterfaceRaw,tkProcVar
  ,
             tkUString,tkUChar,tkHelper,tkFile,tkClassRef,tkPointer
  )
```

Table 75.19: Enumeration values for type TTypeKind

Value	Explanation
tkArray	Array property.
tkAString	Ansistring property.
tkBool	Boolean property.
tkChar	Char property.
tkClass	Class property.
tkClassRef	Class reference type.
tkDynArray	Dynamic array property.
tkEnumeration	Enumeration type property.
tkFile	File type.
tkFloat	Float property.
tkHelper	Helper type.
tkInt64	Int64 property.
tkInteger	Integer property.
tkInterface	Interface property.
tkInterfaceRaw	Raw interface property.
tkLString	Longstring property.
tkMethod	Method property.
tkObject	Object property.
tkPointer	Generic pointer type.
tkProcVar	Procedural variable.
tkQWord	QWord property.
tkRecord	Record property.
tkSet	Set property.
tkSString	Shortstring property.
tkUChar	Unicode character.
tkUnknown	Unknown property type.
tkUString	Unicode string.
tkVariant	Variant property.
tkWChar	Widechar property.
tkWString	Widestring property.

Type of a property or value.

```
TUCS4CharArray = Array[0..$effffff] of UCS4Char
```

Array of UCS4Char (1440) characters.

```
TUnicodeStringManager = record
public
  Wide2AnsiMoveProc : procedure
  (source: PWideChar; var dest: RawByteString;
            cp: TSystemCodePage
  ; len: SizeInt);
  Ansi2WideMoveProc : procedure(source: PChar; cp
  : TSystemCodePage; var dest: widestring;
            len: SizeInt
  );
  UpperWideStringProc : function(const S: WideString) : WideString
  ;
```

```
  LowerWideStringProc : function(const S: WideString) : WideString
  ;
  CompareWideStringProc : function(const s1: WideString; const s2
  : WideString;
           Options: TCompareOptions) : PtrInt;
  CharLengthPCharProc
  : function(const Str: PChar) : PtrInt;
  CodePointLengthProc : function
  (const Str: PChar; MaxLookAead: PtrInt) : PtrInt;
  UpperAnsiStringProc
  : function(const s: ansistring) : ansistring;
  LowerAnsiStringProc
  : function(const s: ansistring) : ansistring;
  CompareStrAnsiStringProc
  : function(const S1: ansistring; const S2: ansistring) : PtrInt;
  CompareTextAnsiStringProc : function(const S1: ansistring; const
  S2: ansistring) : PtrInt;
  StrCompAnsiStringProc : function(S1:
  PChar; S2: PChar) : PtrInt;
  StrICompAnsiStringProc : function(S1
  : PChar; S2: PChar) : PtrInt;
  StrLCompAnsiStringProc : function
  (S1: PChar; S2: PChar; MaxLen: PtrUInt) : PtrInt;
  StrLICompAnsiStringProc
  : function(S1: PChar; S2: PChar; MaxLen: PtrUInt) : PtrInt;
  StrLowerAnsiStringProc
  : function(Str: PChar) : PChar;
  StrUpperAnsiStringProc : function
  (Str: PChar) : PChar;
  ThreadInitProc : procedure;
  ThreadFiniProc
  : procedure;
  Unicode2AnsiMoveProc : procedure(source: PUnicodeChar
  ; var dest: RawByteString;
            cp: TSystemCodePage; len:
  SizeInt);
  Ansi2UnicodeMoveProc : procedure(source: PChar; cp: TSystemCodePage
  ; var dest: unicodestring;
            len: SizeInt);
  UpperUnicodeStringProc
  : function(const S: UnicodeString) : UnicodeString;
  LowerUnicodeStringProc
  : function(const S: UnicodeString) : UnicodeString;
  CompareUnicodeStringProc
  : function(const s1: UnicodeString; const s2: UnicodeString;
  Options: TCompareOptions) : PtrInt;
  GetStandardCodePageProc
  : function(const stdcp: TStandardCodePageEnum) : TSystemCodePage
  ;
end
```

`TUnicodeStringManager` is currently the same as the TUnicodeStringManager (1431) manager record. It performs the same functions: converting Unicode strings to ansistrings and vice-versa, performing uppercase to lowercase transformations and comparing strings.

```
TUnloadLibraryHandler = function(Lib: TLibHandle) : Boolean
```

TUnloadLibraryHandler is the type for the UnloadLibrary call using ansistring names in the TDynLibsManager (1414) dynamic library loading manager.

```
tvararray = record
public
  dimcount : Word;
  flags : Word;
  elementsize
  : LongInt;
  lockcount : LongInt;
  data : pointer;
  bounds : tvararrayboundarray
  ;
end
```

tvararray is a record describing a variant array. It contains some general data, followed by a number of TVarArrayBound (1432) records equal to the number of dimensions in the array (dimcoun).

```
tvararraybound = record
public
  elementcount : LongInt;
  lowbound
  : LongInt;
end
```

tvararraybound is used to describe one dimension in a variant array.

```
tvararrayboundarray = Array[0..0] of tvararraybound
```

array of tvararraybound (1432) records.

```
tvararraycoorarray = Array[0..0] of LongInt
```

Array of variant array coordinates.

```
tvardata = packed record
public
  vtype : tvartype;
case Integer of
  0: (
public
  res1 : Word;
case Integer of
0: (
public
  res2 : Word
  ;
  res3 : Word;
case Word of
varsmallint: (
```

```
public
  vsmallint :
  SmallInt;
);
varinteger: (
public
  vinteger : LongInt;
);
varsingle
  : (
public
  vsingle : single;
);
vardouble: (
public
  vdouble :
  Double;
);
vardate: (
public
  vdate : TDateTime;
);
varcurrency
  : (
public
  vcurrency : currency;
);
varolestr: (
public
  volestr
  : PWideChar;
);
vardispatch: (
public
  vdispatch : pointer;
);
varerror
  : (
public
  verror : HRESULT;
);
varboolean: (
public
  vboolean
  : wordbool;
);
varunknown: (
public
  vunknown : pointer;
);
varustring
  : (
public
  vustring : pointer;
```

```
);
varshortint: (
public
  vshortint
  : ShortInt;
);
varbyte: (
public
  vbyte : Byte;
);
varword: (
public
  vword : Word;
);
varlongword: (
public
  vlongword : DWord;
);
  varint64: (
public
  vint64 : Int64;
);
varqword: (
public
  vqword
  : QWord;
);
varword64: (
public
  vword64 : QWord;
);
varstring:
  (
public
  vstring : pointer;
);
varany: (
public
  vany : pointer
  ;
);
vararray: (
public
  varray : pvararray;
);
varbyref: (
public
  vpointer : pointer;
);
varrecord: (
public
  vrecord : pointer
  ;
  precinfo : pointer;
```

```
);
);
1: (
public
  vlongs : Array[0..2] of
  LongInt;
);
);
1: (
public
  vwords : Array[0..6] of Word;
);
2:
  (
public
  vbytes : Array[0..13] of Byte;
);
end
```

TVarData is a record representation of a variant. It contains the internal structure of a variant and is handled by the various variant handling routines.

```
tvariantmanager = record
public
  vartoint : function(const v: variant
  ) : LongInt;
  vartoint64 : function(const v: variant) : Int64;
  vartoword64 : function(const v: variant) : QWord;
  vartobool : function
  (const v: variant) : Boolean;
  vartoreal : function(const v: variant
  ) : extended;
  vartotdatetime : function(const v: variant) : TDateTime
  ;
  vartocurr : function(const v: variant) : currency;
  vartopstr
  : procedure(var s; const v: variant);
  vartolstr : procedure(var
  s: ansistring; const v: variant);
  vartowstr : procedure(var s:
  widestring; const v: variant);
  vartointf : procedure(var intf:
  IInterface; const v: variant);
  vartodisp : procedure(var disp:
  IDispatch; const v: variant);
  vartodynarray : procedure(var dynarr
  : pointer; const v: variant; typeinfo: pointer);
  varfrombool : procedure
  (var dest: variant; const source: Boolean);
  varfromint : procedure
  (var dest: variant; const source: LongInt;
            const Range
  : LongInt);
  varfromint64 : procedure(var dest: variant; const source
```

```
: Int64);
varfromword64 : procedure(var dest: variant; const source
: QWord);
varfromreal : procedure(var dest: variant; const source
: extended);
varfromtdatetime : procedure(var dest: Variant; const
source: TDateTime);
varfromcurr : procedure(var dest: Variant;
const source: Currency);
varfrompstr : procedure(var dest: variant
; const source: ShortString);
varfromlstr : procedure(var dest:
variant; const source: ansistring);
varfromwstr : procedure(var
dest: variant; const source: WideString);
varfromintf : procedure
(var dest: variant; const source: IInterface);
varfromdisp : procedure
(var dest: variant; const source: IDispatch);
varfromdynarray :
procedure(var dest: variant; const source: pointer; typeinfo: pointer
);
olevarfrompstr : procedure(var dest: olevariant; const source
: shortstring);
olevarfromlstr : procedure(var dest: olevariant
; const source: ansistring);
olevarfromvar : procedure(var dest
: olevariant; const source: variant);
olevarfromint : procedure
(var dest: olevariant; const source: Int64;
          const range
: ShortInt);
varop : procedure(var left: variant; const right: variant
; opcode: tvarop);
cmpop : function(const left: variant; const right
: variant;
         const opcode: tvarop) : Boolean;
varneg
: procedure(var v: variant);
varnot : procedure(var v: variant)
;
varinit : procedure(var v: variant);
varclear : procedure(var
v: variant);
varaddref : procedure(var v: variant);
varcopy
: procedure(var dest: variant; const source: variant);
varcast
: procedure(var dest: variant; const source: variant; vartype: LongInt
);
varcastole : procedure(var dest: variant; const source: variant
; vartype: LongInt);
dispinvoke : procedure(dest: pvardata; var
source: tvardata; calldesc: pcalldesc;
```

```
            params: pointer
  );
  vararrayredim : procedure(var a: variant; highbound: SizeInt
  );
  vararrayget : function(const a: variant; indexcount: SizeInt
  ; indices: PLongint)
            : variant;
  vararrayput : procedure
  (var a: variant; const value: variant; indexcount: SizeInt;
  indices: PLongint);
  writevariant : function(var t: text
  ; const v: variant; width: LongInt) : Pointer;
  write0Variant : function
  (var t: text; const v: Variant) : Pointer;
end
```

`TVariantManager` describes the variant manager as expected by the SetVariantManager (1572) call.

```
tvarop = (opadd,opsubtract,opmultiply,opdivide,opintdivide,opmodulus
  ,
          opshiftleft,opshiftright,opand,opor,opxor,opcompare,opnegate
  ,
          opnot,opcmpeq,opcmpne,opcmplt,opcmple,opcmpgt,opcmpge
  ,oppower)
```

Table 75.20: Enumeration values for type tvarop

Value	Explanation
opadd	Variant operation: Addition.
opand	Variant operation: Binary AND operation.
opcmpeq	Variant operation: Compare equal.
opcmpge	Variant operation: Compare larger than or equal.
opcmpgt	Variant operation: Compare larger than.
opcmple	Variant operation: Compare less than or equal to.
opcmplt	Variant operation: Compare less than.
opcmpne	Variant operation: Compare not equal.
opcompare	Variant operation: Compare.
opdivide	Variant operation: division.
opintdivide	Variant operation: integer divide.
opmodulus	Variant operation: Modulus.
opmultiply	Variant operation: multiplication.
opnegate	Variant operation: negation.
opnot	Variant operation: Binary NOT operation.
opor	Variant operation: Binary OR operation.
oppower	Variant operation: Power.
opshiftleft	Variant operation: Shift left.
opshiftright	Variant operation: Shift right.
opsubtract	Variant operation: Subtraction.
opxor	Variant operation: binary XOR operation.

tvarop describes a variant operation. It is mainly used for the variant manager to implement the various conversions and mathematical operations on a variant.

```
TVarRec = record
case VType : SizeInt of
vtInteger: (
public
  VInteger
  : LongInt;
);
vtBoolean: (
public
  VBoolean : Boolean;
);
vtChar
  : (
public
  VChar : Char;
);
vtWideChar: (
public
  VWideChar : WideChar
  ;
);
vtExtended: (
public
  VExtended : PExtended;
);
vtString: (
  public
  VString : PShortString;
);
vtPointer: (
public
  VPointer
  : Pointer;
);
vtPChar: (
public
  VPChar : PAnsiChar;
);
vtObject
  : (
public
  VObject : TObject;
);
vtClass: (
public
  VClass : TClass
  ;
);
vtPWideChar: (
public
  VPWideChar : PWideChar;
```

```
);
vtAnsiString
  : (
public
  VAnsiString : Pointer;
);
vtCurrency: (
public
  VCurrency
  : PCurrency;
);
vtVariant: (
public
  VVariant : PVariant;
);
vtInterface
  : (
public
  VInterface : Pointer;
);
vtWideString: (
public
  VWideString
  : Pointer;
);
vtInt64: (
public
  VInt64 : PInt64;
);
vtUnicodeString
  : (
public
  VUnicodeString : Pointer;
);
vtQWord: (
public
  VQWord
  : PQWord;
);
end
```

`TVarRec` is a record generated by the compiler for each element in a `array of const` call. The procedure that receives the constant array receives an array of `TVarRec` elements, with lower bound zero and high bound equal to the number of elements in the array minus one (as returned by `High(Args)`)

```
tvartype = Word
```

Type with size of variant type.

```
TWaitForThreadTerminateHandler = function(threadHandle: TThreadID
  ;
                                          TimeoutMs: LongInt)
  : DWord
```

Callback type for thread termination in TThreadManager (1428).

```
TWideStringManager = TUnicodeStringManager
```

`TWideStringManager` contains the definition of the widestring manager.

```
UCS2Char = WideChar
```

UCS2 Unicode character.

```
UCS4Char =
```

UCS Unicode character (unsigned 32 bit word).

```
UCS4String = Array of UCS4Char
```

String of UCS4Char (1440) characters.

```
UInt16 = Word
```

An unsigned 16-bits integer.

```
UInt32 = Cardinal
```

An unsigned 32-bits integer.

```
UInt64 = QWord
```

Unsigned 64-bit integer.

```
UInt8 = Byte
```

An unsigned 8-bits integer.

```
UIntPtr = PtrUInt
```

Alias for PtrUInt (1408) type for compatibility with later Delphi versions.

```
UnicodeChar = WideChar
```

`UnicodeChar` is a single character from a UnicodeString. It equals `WideChar` in all respects.

```
UnicodeString = UnicodeString
```

`UnicodeString` is a string of WideChars. The main difference with WideString is that unicode-string is reference counted, and WideString is not reference counted on Windows.

```
UTF8Char = AnsiChar
```

`UTF8Char` is provided for completeness, a type alias for AnsiChar.

```
UTF8String = ansistring
```

UTF-8 Unicode (Ansi) string.

```
ValReal = Extended
```

`ValReal` is an alias for the largest available floating point type on the architecture the program runs on. On most processors, it should be one of `Double` or `Extended`.

```
ValSInt = LongInt
```

Integer with the same size as the return code of the Val (1598) function.

```
ValSInt = Int64
```

Integer with the same size as the return code of the Val (1598) function.

```
ValUInt = QWord
```

Integer with the same size as the return code of the Val (1598) function.

```
ValUInt = Cardinal
```

Integer with the same size as the return code of the Val (1598) function.

```
WChar = WideChar
```

Wide char (16-bit sized char).

```
WideChar = #$0000..#$FFFF
```

This type is the base unit for all two byte character types, like UnicodeString (1440) and WideString (1441)

```
WideString = WideString
```

`WideString` is an alias for `UnicodeString` on UNIX. On windows, it is a different type which has no reference counting.

```
Word = 0..65535
```

An unsigned 16-bits integer.

75.11.3 Variables

```
argc : LongInt;external 'operatingsystem_parameter_argc'
```

`argc` contains the number of command-line arguments passed to the program by the OS. It is not available on all systems.

```
argv : PPChar;external 'operatingsystem_parameter_argv'
```

argv contains a pointer to a nil-terminated array of null-terminated strings, containing the command-line arguments passed to the program by the OS. It is not available on all systems.

```
DefaultFileSystemCodePage : TSystemCodePage
```

DefaultFileSystemCodePage determines the code page to which file/path names are translated before they are passed to OS API calls, if the RTL uses a single byte OS API for this purpose on the current platform.

This code page is also used for intermediate operations on file paths inside the RTL before making OS API calls.

This variable does not exist in Delphi, and has been introduced in FPC to make it possible to change the value of DefaultSystemCodePage without breaking RTL interfaces with the OS file system API calls.

The initial value of this variable depends on the platform:

- Windows: UTF-8, because the RTL uses UTF-16 OS API calls (so no data is lost in intermediate operations).
- OS X and iOS: UTF-8 (as defined by Apple)
- Unix (excluding OS X and iOS): equals DefaultSystemCodePage (1443). This is because the encoding of file names is undefined on Unix platforms: it is an untyped array of bytes that can be interpreted in any way; Specifically, it is not guaranteed to be valid UTF-8.
- Other platforms: same as DefaultSystemCodePage (1443).

The value of this variable may be changed using the SetMultiByteFileSystemCodePage (1568) procedure.

Remark The Unix/OS X/iOS settings only apply in case the cwstring widestring manager is installed, otherwise DefaultFileSystemCodePage will have the same value as DefaultSystemCodePage after program startup.

```
DefaultRTLFileSystemCodePage : TSystemCodePage
```

DefaultRTLFileSystemCodePage determines the code page to which file/path names are translated before they are returned from RawByteString (1410) file/path RTL routines.

Examples include the file/path names returned by the RawbyteString versions of SysUtils.FindFirst (1362) and GetDir (1501).

The main reason for its existence is to enable the RTL to provide backward compatibility with earlier versions of FPC, as these always returned strings encoded in whatever the OS' single byte API used (normally DefaultSystemCodePage (1443)).

The initial value of this variable depends on the platform:

- Windows: DefaultSystemCodePage, for backward compatibility.
- OS X and iOS: UTF-8, for backward compatibility. It was already always UTF-8 in the past, since that's what the OS file APIs returned, and the data was never converted.
- Other Unixes: DefaultSystemCodePage, for the same reason as DefaultFileSystemCodePage (1442). Setting this to a different value than DefaultFileSystemCodePage is a bad idea on these platforms, since any code page conversion can corrupt these strings as their initial encoding is unknown.

- Other platforms: same as DefaultSystemCodePage.

The value of this variable can be set using the SetMultiByteRTLFileSystemCodePage (1568) call.

```
DefaultSystemCodePage : TSystemCodePage
```

DefaultSystemCodePage is used to determine how CP_ACP is interpreted; it is what the program considers to be the current system codepage.

It is initialized to the default system codepage.

- On windows, this is the result of the GetACP operating call, which returns the Windows ANSI code page.
- On iOS, this is UTF-8
- on other Unixes this will be based on the currently set LANG or LC_CTYPE environment variables. Normally this is UTF-8, but that is not guaranteed to be the case.
- For all other platforms it is set to CP_ACP, as these platforms currently do not support multiple code pages, and are hardcoded to use their OS-specific code page in all cases.

The DefaultSystemCodePage value may be set using SetMultiByteConversionCodePage (1568). That means that it is not a good idea to use its value to determine the real OS "default system code page".

Note that if you change DefaultSystemCodePage, you should call TEncoding.FreeEncodings to free the encodings that were created. They will be recreated with the correct (new) DefaultSystemCodePage.

```
DefaultUnicodeCodePage : TSystemCodePage
```

DefaultUnicodeCodePage is the unicode code page for a new unicode string. On most platforms, this is CP_UTF16 (1370).

```
DispCallByIDProc : CodePointer
```

VarDispProc is called by the compiler if it needs to perform an interface call from a variant which contains a dispatch interface. For instance, the following call:

```
Var
  V : OleVariant;
begin
  (V as IWord).OpenDocument('c:\temp\mydoc.doc');
end;
```

where IWord is a dispatch interface is encoded by the compiler and passed to DispCallByIDProc.

This pointer must be set by a routine that calls the OS COM handling routines.

```
envp : PPChar;external 'operatingsystem_parameter_envp'
```

envp contains a pointer to a nil-terminated array of null-terminated strings, containing the environment variables passed to the program by the OS. It is not available on all systems.

```
ErrOutput : Text
```

ErrOutput is provided for Delphi compatibility.

```
ExitCode : LongInt;public 'operatingsystem_result'
```

Exit code for the program, will be communicated to the OS on exit.

```
FirstDotAtFileNameStartIsExtension : Boolean = False
```

FirstDotAtFileNameStartIsExtension determines what happens if a filename starts with a dot (.) character. If True, then the whole file name will be treated as extension. If False, then the extension is empty.

```
InOutRes : Word
```

InOutRes contains the result of the last I/O operation using one of the file I/O routines. When I/O checks are enabled, this variable is checked and if it is nonzero, a runtime error is raised.

Although it is a variable for historical reasons, it is not meant to be written to by user code, outside of routines that implement some form of file I/O.

```
Input : Text
```

Standard input text file.

```
IsConsole : Boolean;public 'operatingsystem_isconsole' = False
```

True for console applications, False for GUI applications.

```
IsLibrary : Boolean;public 'operatingsystem_islibrary' = False
```

True if the current module is a library. Otherwise module is an executable.

```
mem : Array[0..$7fffffff-1] of Byte
```

mem is an array of bytes, representing the computer's memory. This array is available only when compiling for the Dos Go32V2 target. It's use is not recommended, and it is not even available on other platforms.

```
meml : Array[0..($7fffffffdivsizeof(longint))-1] of LongInt
```

memw is an array of longints, representing the computer's memory as 32-bit signed integers. This array is available only when compiling for the Dos Go32V2 target. It's use is not recommended, and it is not even available on other platforms.

```
memw : Array[0..($7fffffffdivsizeof(word))-1] of Word
```

memw is an array of words, representing the computer's memory as 2-byte words. This array is available only when compiling for the Dos Go32V2 target It's use is not recommended, and it is not even available on other platforms.

```
NoErrMsg : Boolean = Falseplatform
```

Unused, for Delphi compatibility.

```
Output : Text
```

Standard output text file.

```
RandSeed : Cardinal
```

Seed for Random (1546) function.

```
ReturnNilIfGrowHeapFails : Boolean
```

`ReturnNilIfGrowHeapFails` describes what happens if there is no more memory available from the operating system. if set to `True` the memory manager will return `Nil`. If set to `False` then a run-time error will occur.

```
softfloat_exception_flags : TFPUExceptionMask
```

Current soft float exception flags.

```
softfloat_exception_mask : TFPUExceptionMask
```

Current soft float exception mask.

```
softfloat_rounding_mode : TFPURoundingMode
```

`softfloat_rounding_mode` determines how the software floating-point emulation routines do the rounding. The value can be one of the following:

float_round_nearest_even Round to nearest even number.

float_round_down Round down.

float_round_up Round up.

float_round_to_zero Round in the direction of zero (down for positive, up for negative).

```
StackBottom : Pointer
```

Current stack bottom.

```
StackLength : SizeUInt
```

Maximum stack length.

```
StdErr : Text
```

Standard diagnostic output text file.

```
StdOut : Text
```

Alias for Output (1445).

`ThreadID : TThreadID`

Current Thread ID.

`UTF8CompareLocale : TSystemCodePage`

`UTF8CompareLocale` is currently present for Delphi compatibility only, it is not used in FPC code.

`widestringmanager : TUnicodeStringManager`

Contains the current widestring manager. Do not use directly.

`WriteErrorsToStdErr : Boolean = True`

`WriteErrorsToStdErr` can be set to `True` to write error messages (run-time errors, exceptions) to `StdErr` instead of to standard output. This is the default behaviour. When set to `False`, error message will be written to standard output.

75.12 Procedures and functions

75.12.1 Abs

Synopsis: Calculate absolute value.

Declaration:
```
function Abs(l: LongInt) : LongInt
function Abs(l: Int64) : Int64
function Abs(d: ValReal) : ValReal
```

Visibility: default

Description: `Abs` returns the absolute value of a variable. The result of the function has the same type as its argument, which can be any numerical type.

Errors: None.

See also: Round (1557)

Listing: ./examples/refex/ex1.pp

```
Program Example1;

{ Program to demonstrate the Abs function. }

Var
  r : real;
  i : integer;

begin
  r:=abs(-1.0);    { r:=1.0 }
  i:=abs(-21);     { i:=21 }
end.
```

75.12.2 AbstractError

Synopsis: Generate an abstract error.

Declaration: `procedure AbstractError`

Visibility: default

Description: `AbstractError` generates an abstract error (run-time error 211). If the AbstractErrorProc (1368) constant is set, it will be called instead.

Errors: This routine causes a run-time error 211.

See also: AbstractErrorProc (1368)

75.12.3 AcquireExceptionObject

Synopsis: Obtain a reference to the current exception object.

Declaration: `function AcquireExceptionObject : Pointer`

Visibility: default

Description: `AcquireExceptionObject` returns the current exception object. It raises the reference count of the exception object, so it will not be freed. Calling this method is only valid within an except block.

The effect of this function is countered by re-raising an exception via `raise;`

To make sure that the exception object is released when it is no longer needed, ReleaseExceptionObject (1551) must be called when the reference is no longer needed.

Errors: If there is no current exception, a run-time error 231 will occur.

See also: ReleaseExceptionObject (1551)

75.12.4 add(variant,variant):variant

Synopsis: Implement addition `(+)` operation on variants.

Declaration: `operator +(const op1: variant; const op2: variant) : variant`

Visibility: default

Description: The implementation of the addition + operation is delegated to the variant manager with operation `opadd`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator -(variant, variant): variant (1362)

75.12.5 AddExitProc

Synopsis: Add an exit procedure to the exit procedure chain.

Declaration: `procedure AddExitProc(Proc: TProcedure)`

Visibility: default

Description: `AddExitProc` adds `Proc` to the exit procedure chain. At program exit, all procedures added in this way will be called in reverse order.

Errors: None.

See also: ExitProc (1372)

75.12.6 Addr

Synopsis: Return address of a variable.

Declaration: `function Addr(X: TAnytype) : Pointer`

Visibility: default

Description: `Addr` returns a pointer to its argument, which can be any type, or a function or procedure name. The returned pointer isn't typed. The same result can be obtained by the `@` operator, which can return a typed pointer (see the programmer's guide).

Errors: None

See also: SizeOf (1574)

Listing: ./examples/refex/ex2.pp

```
Program Example2;

{ Program to demonstrate the Addr function. }

Const Zero : integer = 0;

Var p : pointer;
    i : Integer;

begin
  p:=Addr(p);      { P points to itself }
  p:=Addr(I);      { P points to I }
  p:=Addr(Zero);   { P points to 'Zero' }
end.
```

75.12.7 Align

Synopsis: Return aligned version of an address.

Declaration:
```
function Align(Addr: PtrUInt; Alignment: PtrUInt) : PtrUInt
function Align(Addr: Pointer; Alignment: PtrUInt) : Pointer
```

Visibility: default

Description: `Align` returns `Address`, aligned to `Alignment` bytes.

Errors: None.

75.12.8 AllocMem

Synopsis: Allocate and clear memory.

Declaration: `function AllocMem(Size: PtrUInt) : pointer`

Visibility: default

Description: `AllocMem` calls getmem GetMem (1502), and clears the allocated memory, i.e. the allocated memory is filled with `Size` zero bytes.

See also: GetMem (1502)

75.12.9 AnsiToUtf8

Synopsis: Convert ansi string to UTF-8 string.

Declaration: `function AnsiToUtf8(const s: RawByteString) : RawByteString`

Visibility: default

Description: `AnsiToUtf8` converts the ansistring `S` to a UTF-8 format, that is, it converts the string from whatever codepage is currently in use, to UTF-8.

The current codepage is fetched from the system, if internationalization support is enabled. It can be UTF-8, in which case the function simply returns `S`.

Errors: None.

See also: Utf8toAnsi (1597)

75.12.10 Append

Synopsis: Open a file in append mode.

Declaration: `procedure Append(var t: Text)`

Visibility: default

Description: `Append` opens an existing file in append mode. Any data written to `F` will be appended to the file. Only text files can be opened in append mode. After a call to `Append`, the file `F` becomes write-only. File sharing is not taken into account when calling `Append`.

Errors: If the file doesn't exist when appending, a run-time error will be generated. This behaviour has changed on Windows and Linux platforms, where in versions prior to 1.0.6, the file would be created in append mode.

See also: Rewrite (1553), Close (1465), Reset (1552)

Listing: ./examples/refex/ex3.pp

```
Program Example3;

{ Program to demonstrate the Append function. }

Var f : text;

begin
  Assign (f,'test.txt');
  Rewrite (f);            { file is opened for write, and emptied }
```

```
  WriteLn (F,'This is the first line of text.txt');
  close (f);
  Append(f);              { file is opened for write, but NOT emptied.
                            any text written to it is appended.}
  WriteLn (f,'This is the second line of text.txt');
  close (f);
end.
```

75.12.11 ArcTan

Synopsis: Calculate inverse tangent.

Declaration: `function ArcTan(d: ValReal) : ValReal`

Visibility: default

Description: `Arctan` returns the Arctangent of `X`, which can be any Real type. The resulting angle is in radial units.

Errors: None

See also: Sin (1573), Cos (1474)

Listing: ./examples/refex/ex4.pp

```
Program Example4;

{ Program to demonstrate the ArcTan function. }

Var R : Real;

begin
  R:=ArcTan(0);       { R:=0 }
  R:=ArcTan(1)/pi;    { R:=0.25 }
end.
```

75.12.12 ArrayStringToPPchar

Synopsis: Concert an array of string to an array of null-terminated strings.

Declaration: `function ArrayStringToPPchar(const S: Array of AnsiString;`
`reserveentries: LongInt) : PPChar`

Visibility: default

Description: `ArrayStringToPPchar` creates an array of null-terminated strings that point to strings which are the same as the strings in the array `S`. The function returns a pointer to this array. The array and the strings it contains must be disposed of after being used, because it they are allocated on the heap.

The `ReserveEntries` parameter tells `ArrayStringToPPchar` to allocate room at the end of the array for another `ReserveEntries` entries.

Errors: If not enough memory is available, an error may occur.

See also: StringToPPChar (1580)

75.12.13 Assert

Synopsis: Check validity of a given condition.

Declaration:
```
procedure Assert(Expr: Boolean)
procedure Assert(Expr: Boolean; const Msg: string)
```

Visibility: default

Description: With assertions on, `Assert` tests if `expr` is false, and if so, aborts the application with a Runtime error 227 and an optional error message in `msg`. If `expr` is true, program execution continues normally. If assertions are not enabled at compile time, this routine does nothing, and no code is generated for the `Assert` call. Enabling and disabling assertions at compile time is done via the `\$C` or `\$ASSERTIONS` compiler switches. These are local switches. The default behavior of the assert call can be changed by setting a new handler in the `AssertErrorProc` variable. Sysutils overrides the default handler to raise a `EAssertionFailed` exception.

Errors: None.

See also: Halt (1509), Runerror (1560)

75.12.14 Assign

Synopsis: Assign a name to a file.

Declaration:
```
procedure Assign(out f: File; const Name: ShortString)
procedure Assign(out f: File; const p: PAnsiChar)
procedure Assign(out f: File; const c: AnsiChar)
procedure Assign(out f: File; const Name: UnicodeString)
procedure Assign(out f: File; const Name: RawByteString)
procedure Assign(out f: TypedFile; const Name: shortstring)
procedure Assign(out f: TypedFile; const p: PAnsiChar)
procedure Assign(out f: TypedFile; const c: AnsiChar)
procedure Assign(out f: TypedFile; const Name: unicodestring)
procedure Assign(out f: TypedFile; const Name: RawByteString)
procedure Assign(out t: Text; const s: shortstring)
procedure Assign(out t: Text; const p: PAnsiChar)
procedure Assign(out t: Text; const c: AnsiChar)
procedure Assign(out t: Text; const s: unicodestring)
procedure Assign(out t: Text; const s: RawByteString)
```

Visibility: default

Description: `Assign` assigns a name to `F`, which can be any file type. This call doesn't open the file, it just assigns a name to a file variable, and marks the file as closed.

Note that the filename (including path) can be only 255 characters long.

Errors: None.

See also: Reset (1552), Rewrite (1553), Append (1449)

Listing: ./examples/refex/ex5.pp

```
Program Example5;

{ Program to demonstrate the Assign function. }

Var F : text;
```

```
begin
  Assign (F,'');
  Rewrite (f);
  { The following can be put in any file by redirecting it
    from the command line.}
  Writeln (f,'This goes to standard output !');
  Close (f);
  Assign (F,'Test.txt');
  rewrite (f);
  writeln (f,'This doesn''t go to standard output !');
  close (f);
end.
```

75.12.15 assign(Comp):olevariant

Synopsis: Assign a comp-precision float to an ole-variant.

Declaration: `operator :=(const source: Comp) : olevariant`

Visibility: default

75.12.16 assign(Comp):variant

Synopsis: Assign a comp-precision float to a variant.

Declaration: `operator :=(const source: Comp) : variant`

Visibility: default

Description: The resulting variant is a double-precision value.

75.12.17 assign(extended):olevariant

Synopsis: Assign an extended-precision float to an ole-variant.

Declaration: `operator :=(const source: extended) : olevariant`

Visibility: default

75.12.18 assign(extended):variant

Synopsis: Assign an extended-precision float to a variant.

Declaration: `operator :=(const source: extended) : variant`

Visibility: default

Description: The resulting variant is a double-precision value

75.12.19 assign(NativeInt):variant

Declaration: `operator :=(const source: NativeInt) : variant`

Visibility: default

75.12.20 assign(NativeUInt):variant

Declaration: `operator :=(const source: NativeUInt) : variant`

Visibility: default

75.12.21 assign(olevariant):Comp

Synopsis: Assign an ole- variant to a comp-precision float.

Declaration: `operator :=(const source: olevariant) : Comp`

Visibility: default

75.12.22 assign(olevariant):extended

Synopsis: Assign an ole- variant to an extended-precision float.

Declaration: `operator :=(const source: olevariant) : extended`

Visibility: default

75.12.23 assign(olevariant):Real

Synopsis: Assign an ole- variant to a real-precision float.

Declaration: `operator :=(const source: olevariant) : Real`

Visibility: default

75.12.24 assign(olevariant):single

Synopsis: Assign an ole- variant to a single-precision float.

Declaration: `operator :=(const source: olevariant) : single`

Visibility: default

75.12.25 assign(olevariant):UnicodeString

Synopsis: Assign an ole-variant to a unicode string.

Declaration: `operator :=(const source: olevariant) : UnicodeString`

Visibility: default

75.12.26 assign(Real):olevariant

Synopsis: Assign a real-precision float to an ole-variant.

Declaration: `operator :=(const source: Real) : olevariant`

Visibility: default

75.12.27 assign(Real):variant

Synopsis: Assign a real-precision float to a variant.

Declaration: `operator :=(const source: Real) : variant`

Visibility: default

Description: The resulting variant is a double-precision value.

75.12.28 assign(Real48):extended

Synopsis:

Declaration: `operator :=(b: Real48) : extended`

Visibility: default

75.12.29 assign(single):olevariant

Synopsis: Assign a single-precision float to an ole-variant.

Declaration: `operator :=(const source: single) : olevariant`

Visibility: default

75.12.30 assign(single):variant

Synopsis: Assign a single-precision float to a variant.

Declaration: `operator :=(const source: single) : variant`

Visibility: default

Description: The resulting variant is a double-precision value

75.12.31 assign(UCS4String):variant

Synopsis: Assign UCS4String to a variant, performing the necessary conversions.

Declaration: `operator :=(const source: UCS4String) : variant`

Visibility: default

Description: The resulting variant is a widestring.

75.12.32 assign(UnicodeString):olevariant

Synopsis: Assign a unicodestring to an ole-variant.

Declaration: `operator :=(const source: UnicodeString) : olevariant`

Visibility: default

75.12.33 assign(UnicodeString):variant

Synopsis: Assign UnicodeString to a variant, performing the necessary conversions.

Declaration: `operator :=(const source: UnicodeString) : variant`

Visibility: default

Description: The resulting variant is a widestring.

75.12.34 assign(UTF8String):variant

Synopsis: Assign an UTF-8 string to a variant, performing the necessary conversions.

Declaration: `operator :=(const source: UTF8String) : variant`

Visibility: default

Description: The resulting variant is a widestring.

75.12.35 assign(variant):Comp

Synopsis: Assign a variant to a comp-precision float.

Declaration: `operator :=(const source: variant) : Comp`

Visibility: default

75.12.36 assign(variant):extended

Synopsis: Assign a variant to a extended-precision float.

Declaration: `operator :=(const source: variant) : extended`

Visibility: default

75.12.37 assign(variant):NativeInt

Declaration: `operator :=(const source: variant) : NativeInt`

Visibility: default

75.12.38 assign(variant):NativeUInt

Declaration: `operator :=(const source: variant) : NativeUInt`

Visibility: default

75.12.39 assign(variant):Real

Synopsis: Assign a variant to a real-precision float.

Declaration: `operator :=(const source: variant) : Real`

Visibility: default

75.12.40 assign(variant):single

Synopsis: Assign a variant to a single-precision float.

Declaration: `operator :=(const source: variant) : single`

Visibility: default

75.12.41 assign(variant):unicodestring

Synopsis: Assign a variant to a unicodestring.

Declaration: `operator :=(const source: variant) : unicodestring`

Visibility: default

75.12.42 assign(variant):UTF8String

Synopsis: Assign a variant to an UTF8String.

Declaration: `operator :=(const source: variant) : UTF8String`

Visibility: default

75.12.43 Assigned

Synopsis: Check if a pointer is valid.

Declaration: `function Assigned(P: Pointer) : Boolean`

Visibility: default

Description: `Assigned` returns `True` if `P` is non-nil and returns `False` of `P` is nil. The main use of Assigned is that Procedural variables, method variables and class-type variables also can be passed to `Assigned`.

Errors: None

See also: New (1537)

Listing: ./examples/refex/ex96.pp

```
Program Example96;

{ Program to demonstrate the Assigned function. }

Var P : Pointer;

begin
  If Not Assigned(P) then
    WriteIn ('Pointer is initially NIL');
  P:=@P;
  If Not Assigned(P) then
    WriteIn('Internal inconsistency')
  else
    WriteIn('All is well in FPC')
end.
```

75.12.44 BasicEventCreate

Synopsis: Obsolete. Don't use.

Declaration:
```
function BasicEventCreate(EventAttributes: Pointer;
                          AManualReset: Boolean; InitialState: Boolean;
                          const Name: ansistring) : PEventState
```

Visibility: default

Description: `BasicEventCreate` is obsolete, use RTLEventCreate (1559) instead.

See also: RTLEventCreate (1559)

75.12.45 BasicEventDestroy

Synopsis: Obsolete. Don't use.

Declaration:
```
procedure BasicEventDestroy(state: PEventState)
```

Visibility: default

Description: `basiceventdestroy` is obsolete. Use RTLEventDestroy (1559) instead.

See also: RTLEventDestroy (1559)

75.12.46 BasicEventResetEvent

Synopsis: Obsolete. Don't use.

Declaration:
```
procedure BasicEventResetEvent(state: PEventState)
```

Visibility: default

Description: `basiceventResetEvent` is obsolete. Use RTLEventResetEvent (1559) instead.

See also: RTLEventResetEvent (1559)

75.12.47 BasicEventSetEvent

Synopsis: Obsolete. Don't use.

Declaration:
```
procedure BasicEventSetEvent(state: PEventState)
```

Visibility: default

Description: `basiceventSetEvent` is obsolete. Use RTLEventSetEvent (1560) instead.

See also: RTLEventSetEvent (1560)

75.12.48 BasicEventWaitFor

Synopsis: Obsolete. Don't use.

Declaration:
```
function BasicEventWaitFor(Timeout: Cardinal; state: PEventState)
                           : LongInt
```

Visibility: default

Description: `basiceventwaitfor` is obsolete. Use RTLEventWaitFor (1560) instead.

See also: RTLEventWaitFor (1560)

75.12.49 BeginThread

Synopsis: Start a new thread.

Declaration:
```
function BeginThread(sa: Pointer; stacksize: SizeUInt;
                     ThreadFunction: TThreadFunc; p: pointer;
                     creationFlags: DWord; var ThreadId: TThreadID)
                      : TThreadID
function BeginThread(ThreadFunction: TThreadFunc) : TThreadID
function BeginThread(ThreadFunction: TThreadFunc; p: pointer)
                      : TThreadID
function BeginThread(ThreadFunction: TThreadFunc; p: pointer;
                     var ThreadId: TThreadID) : TThreadID
function BeginThread(ThreadFunction: TThreadFunc; p: pointer;
                     var ThreadId: TThreadID; const stacksize: SizeUInt)
                      : TThreadID
```

Visibility: default

Description: `BeginThread` starts a new thread and executes `ThreadFunction` in the new thread. If `P` is specified, then it is passed to `ThreadFunction`. If `ThreadId` is specified, it is filled with the thread ID of the newly started thread. If `StackSize` is specified, it is set as the stack size for the new thread. If none is specified, a default stack size of 4MiB is used.

The function returns the thread handle (or ID, on some other operating systems like Linux or OS/2) on success, or 0 if an error occurred. Note that the thread ID and handle are the same on Unix processes, and that the thread ID and thread handle are different on windows systems.

Errors: On error, the value "0" is returned.

See also: EndThread (1483)

75.12.50 BEtoN

Synopsis: Convert Big Endian-ordered integer to Native-ordered integer.

Declaration:
```
function BEtoN(const AValue: SmallInt) : SmallInt
function BEtoN(const AValue: Word) : Word
function BEtoN(const AValue: LongInt) : LongInt
function BEtoN(const AValue: DWord) : DWord
function BEtoN(const AValue: Int64) : Int64
function BEtoN(const AValue: QWord) : QWord
```

Visibility: default

Description: `BEToN` will rearrange the bytes in a Big-Endian number to the native order for the current processor. That is, for a big-endian processor, it will do nothing, and for a little-endian processor, it will invert the order of the bytes.

See also: LEtoN (1529), NtoBE (1538), NtoLE (1538)

75.12.51 BinStr

Synopsis: Convert integer to string with binary representation.

Declaration:
```
function BinStr(Val: LongInt; cnt: Byte) : shortstring
function BinStr(Val: Int64; cnt: Byte) : shortstring
function BinStr(Val: QWord; cnt: Byte) : shortstring
```

Visibility: default

Description: `BinStr` returns a string with the binary representation of `Value`. The string has at most `cnt` characters. (i.e. only the `cnt` rightmost bits are taken into account) To have a complete representation of any longint-type value, 32 bits are needed, i.e. `cnt=32`

Errors: None.

See also: Str (1577), Val (1598), HexStr (1509), OctStr (1539)

Listing: ./examples/refex/ex82.pp

```
Program example82;

{ Program to demonstrate the BinStr function }

Const Value = 45678;

Var I : longint;

begin
  For I:=8 to 20 do
    WriteIn (BinStr(Value,I):20);
end.
```

75.12.52 BlockRead

Synopsis: Read data from an untyped file into memory.

Declaration:
```
procedure BlockRead(var f: File; var Buf; count: Int64;
                    var Result: Int64)
procedure BlockRead(var f: File; var Buf; count: LongInt;
                    var Result: LongInt)
procedure BlockRead(var f: File; var Buf; count: Cardinal;
                    var Result: Cardinal)
procedure BlockRead(var f: File; var Buf; count: Word; var Result: Word)
procedure BlockRead(var f: File; var Buf; count: Word;
                    var Result: Integer)
procedure BlockRead(var f: File; var Buf; count: Int64)
```

Visibility: default

Description: `Blockread` reads `count` or less records from file `F`. A record is a block of bytes with size specified by the Rewrite (1553) or Reset (1552) statement. The result is placed in `Buffer`, which must contain enough room for `Count` records. The function cannot read partial records. If `Result` is specified, it contains the number of records actually read. If `Result` isn't specified, and less than `Count` records were read, a run-time error is generated. This behavior can be controlled by the `{$I}` switch.

Errors: Depending on the state of the `{$I}` switch, a runtime error can be generated if there is an error. In the `{$I-}` state, use `IOResult` to check for errors.

See also: Blockwrite (1460), Close (1465), Reset (1552), Assign (1451)

Listing: ./examples/refex/ex6.pp

```
Program Example6;

{ Program to demonstrate the BlockRead and BlockWrite functions. }

Var Fin, fout : File;
    NumRead, NumWritten : Word;
    Buf : Array[1..2048] of byte;
    Total : Longint;

begin
  Assign (Fin, Paramstr(1));
  Assign (Fout,Paramstr(2));
  Reset (Fin,1);
  Rewrite (Fout,1);
  Total:=0;
  Repeat
    BlockRead (Fin,buf,Sizeof(buf),NumRead);
    BlockWrite (Fout,Buf,NumRead,NumWritten);
    inc(Total,NumWritten);
  Until (NumRead=0) or (NumWritten<>NumRead);
  Write ('Copied ',Total,' bytes from file ',paramstr(1));
  WriteIn (' to file ',paramstr(2));
  close(fin);
  close(fout);
end.
```

75.12.53 BlockWrite

Synopsis: Write data from memory to an untyped file.

Declaration:
```
procedure BlockWrite(var f: File; const Buf; Count: Int64;
                     var Result: Int64)
procedure BlockWrite(var f: File; const Buf; Count: LongInt;
                     var Result: LongInt)
procedure BlockWrite(var f: File; const Buf; Count: Cardinal;
                     var Result: Cardinal)
procedure BlockWrite(var f: File; const Buf; Count: Word;
                     var Result: Word)
procedure BlockWrite(var f: File; const Buf; Count: Word;
                     var Result: Integer)
procedure BlockWrite(var f: File; const Buf; Count: LongInt)
```

Visibility: default

Description: `BlockWrite` writes `count` records from `buffer` to the file `F`.A record is a block of bytes with size specified by the Rewrite (1553) or Reset (1552) statement. If the records couldn't be written to disk, a run-time error is generated. This behavior can be controlled by the `{$I}` switch.

Errors: Depending on the state of the `{$I}` switch, a runtime error can be generated if there is an error. In the `{$I-}` state, use `IOResult` to check for errors.

See also: Blockread (1459), Close (1465), Rewrite (1553), Assign (1451)

75.12.54 Break

Synopsis: Exit current loop construct.

Declaration: `procedure Break`

Visibility: default

Description: `Break` jumps to the statement following the end of the current repetitive statement. The code between the `Break` call and the end of the repetitive statement is skipped. The condition of the repetitive statement is NOT evaluated.

This can be used with `For`, var{repeat} and `While` statements.

Note that although `Break` is a compiler intrinsic (i.e. is treated specially) it is defined as a procedure in the system unit, and hence can be redefined.

Errors: None.

See also: Continue (1472), Exit (1488)

Listing: ./examples/refex/ex87.pp

```
Program Example87;

{ Program to demonstrate the Break function. }

Var I : longint;

begin
  I:=0;
  While I<10 Do
    begin
    Inc(I);
    If I>5 Then
      Break;
    WriteIn (i);
    end;
  I:=0;
  Repeat
    Inc(I);
    If I>5 Then
      Break;
    WriteIn (i);
  Until I>=10;
  For I:=1 to 10 do
    begin
    If I>5 Then
      Break;
    WriteIn (i);
    end;
end.
```

Listing: ./examples/refex/ex121.pp

```
{
  Example 121:
  Continue, break and exit are system procedures.
  They can be redefined
}

procedure continue;

begin
```

```
  WriteLn('Continue');
end;

Procedure Exit;

begin
  WriteLn('exit');
end;

Procedure Break;

begin
  WriteLn('Break');
end;

begin
  Repeat
    Continue;
    Break;
    exit;
  Until True;
end.
```

75.12.55 BsfByte

Synopsis: Return the position of the rightmost set bit in an 8-bit value.

Declaration: `function BsfByte(const AValue: Byte) : Byte`

Visibility: default

Description: `BsfByte` scans the byte `AValue`, starting at position 0 (rightmost position) and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit.

When the input is 0, the result is 255 (unsigned equivalent of -1).

See also: BsrByte (1463), BsfWord (1463), BsfDWord (1462), BsfQWord (1462)

75.12.56 BsfDWord

Synopsis: Return the position of the rightmost set bit in a 32-bit value.

Declaration: `function BsfDWord(const AValue: DWord) : Cardinal`

Visibility: default

Description: `BsfDWord` scans the DWord `AValue`, starting at position 0 (rightmost position) , and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit.

When the input is 0, the result is 255 (unsigned equivalent of -1).

See also: BsfByte (1462), BsfWord (1463), BsrDWord (1463), BsfQWord (1462)

75.12.57 BsfQWord

Synopsis: Return the position of the rightmost set bit in a 64-bit value.

Declaration: `function BsfQWord(const AValue: QWord) : Cardinal`

Visibility: default

Description: `BsfQWord` scans the QWord `AValue`, starting at position 0 (rightmost position) , and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit.

When the input is 0, the result is 255 (unsigned equivalent of -1).

See also: BsfByte (1462), BsfWord (1463), BsfDWord (1462), BsrQWord (1464)

75.12.58 BsfWord

Synopsis: Return the position of the rightmost set bit in a 16-bit value.

Declaration: `function BsfWord(const AValue: Word) : Cardinal`

Visibility: default

Description: `BsfWord` scans the word `AValue`, starting at position 0 (rightmost position) , and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit.

When the input is 0, the result is 255 (unsigned equivalent of -1).

See also: BsfByte (1462), BsrWord (1464), BsfDWord (1462), BsfQWord (1462)

75.12.59 BsrByte

Synopsis: Return the position of the leftmost set bit in an 8-bit value.

Declaration: `function BsrByte(const AValue: Byte) : Byte`

Visibility: default

Description: `BsfByte` scans the byte `AValue`, starting at the leftmost position and working towards position 0, and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit.

When the input is 0, the result is 255 (unsigned equivalent of -1).

See also: BsfByte (1462), BsrWord (1464), BsrDWord (1463), BsrQWord (1464)

75.12.60 BsrDWord

Synopsis: Return the position of the leftmost set bit in a 32-bit value.

Declaration: `function BsrDWord(const AValue: DWord) : Cardinal`

Visibility: default

Description: `BsrDWord` scans the DWord `AValue`, starting at the leftmost position and working towards position 0, and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit.

When the input is 0, the result is 255 (unsigned equivalent of -1).

See also: BsrByte (1463), BsrWord (1464), BsfDWord (1462), BsrQWord (1464)

75.12.61 BsrQWord

Synopsis: Return the position of the leftmost set bit in a 64-bit value.

Declaration:
```
function BsrQWord(const AValue: QWord) : Cardinal
```

Visibility: default

Description: `BsfQWord` scans the QWord `AValue`, starting at the leftmost position and working towards position 0, and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit.

When the input is 0, the result is 255 (unsigned equivalent of -1).

See also: BsfByte (1462), BsfWord (1463), BsfDWord (1462), BsrQWord (1464)

75.12.62 BsrWord

Synopsis: Return the position of the leftmost set bit in a 16-bit value.

Declaration:
```
function BsrWord(const AValue: Word) : Cardinal
```

Visibility: default

Description: `BsrWord` scans the word `AValue`, starting at the leftmost position and working towards position 0, and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit.

When the input is 0, the result is 255 (unsigned equivalent of -1).

See also: BsrByte (1463), BsfWord (1463), BsrDWord (1463), BsrQWord (1464)

75.12.63 CaptureBacktrace

Synopsis: Return stack trace.

Declaration:
```
function CaptureBacktrace(skipframes: SizeInt; count: SizeInt;
                          frames: PCodePointer) : SizeInt
```

Visibility: default

Description: `CaptureBacktrace` will fill the array pointed to by `frames` with the addresses of a backtrace. It will skip `skipframes` frames, and will write at most `count` addresses. `Frames` must point to enough memory to hold the stacktrace, which is `count*sizeof(codepointer)` bytes.

See also: Get_pc_addr (1508), get_caller_stackinfo (1507), get_caller_addr (1507), get_caller_frame (1507)

75.12.64 ChDir

Synopsis: Change current working directory.

Declaration:
```
procedure ChDir(const s: shortstring);  Overload
procedure ChDir(const s: RawByteString);  Overload
procedure ChDir(const s: unicodestring);  Overload
```

Visibility: default

Description: `Chdir` changes the working directory of the process to `S`.

Errors: Depending on the state of the `{$I}` switch, a runtime error can be generated if there is an error. In the `{$I-}` state, use `IOResult` to check for errors.

See also: Mkdir (1535), Rmdir (1554)

Listing: ./examples/refex/ex7.pp

```
Program Example7;

{ Program to demonstrate the ChDir function. }

begin
  {$I-}
  ChDir (ParamStr(1));
  if IOresult<>0 then
    WriteIn ('Cannot change to directory : ',paramstr (1));
end.
```

75.12.65 Chr

Synopsis: Convert byte value to character value.

Declaration:
```
function Chr(b: Byte) : Char
```

Visibility: default

Description: `Chr` returns the character which has ASCII value `X`.

Historical note:

Originally, Pascal did not have typecasts and chr was a necessary function in order to do certain operations on ASCII values of characters. With the arrival of typecasting a generic approach became possible, making chr mostly obsolete. However, chr is not considered deprecated and remains in wide use today.

Errors: None.

See also: Ord (1540), Str (1577)

Listing: ./examples/refex/ex8.pp

```
Program Example8;

{ Program to demonstrate the Chr function. }

begin
  Write (chr(10),chr(13)); { The same effect as WriteIn; }
end.
```

75.12.66 Close

Synopsis: Close a file.

Declaration:
```
procedure Close(var f: File)
procedure Close(var t: Text)
```

Visibility: default

Description: `Close` flushes the buffer of the file `F` and closes `F`. After a call to `Close`, data can no longer be read from or written to `F`. To reopen a file closed with `Close`, it isn't necessary to assign the file again. A call to Reset (1552) or Rewrite (1553) is sufficient.

Errors: Depending on the state of the `{$I}` switch, a runtime error can be generated if there is an error. In the `{$I-}` state, use `IOResult` to check for errors.

See also: Assign (1451), Reset (1552), Rewrite (1553), Flush (1497)

Listing: ./examples/refex/ex9.pp

```
Program Example9;

{ Program to demonstrate the Close function. }

Var F : text;

begin
 Assign (f,'Test.txt');
 ReWrite (F);
 WriteIn (F,'Some text written to Test.txt');
 close (f); { Flushes contents of buffer to disk,
              closes the file. Omitting this may
              cause data NOT to be written to disk.}
end.
```

75.12.67 CloseThread

Synopsis: Close a thread and free up resources used by the thread.

Declaration: `function CloseThread(threadHandle: TThreadID) : DWord`

Visibility: default

Description: `CloseThread` must be called on any thread started with BeginThread (1458). It must be called after the thread has ended (either by exiting the thread function or after calling EndThread (1483)).

Errors: If no threadmanager is installed, an exception may be raised or runtime error 232 may occur if no exceptions are used.

See also: BeginThread (1458), EndThread (1483)

75.12.68 CompareByte

Synopsis: Compare 2 memory buffers byte per byte.

Declaration: `function CompareByte(const buf1; const buf2; len: SizeInt) : SizeInt`

Visibility: default

Description: `CompareByte` compares two memory regions `buf1`,`buf2` on a byte-per-byte basis for a total of `len` bytes.

The function returns one of the following values:

less than 0 if `buf1` and `buf2` contain different bytes in the first `len` bytes, and the first such byte is smaller in `buf1` than the byte at the same position in `buf2`.

0 if the first `len` bytes in `buf1` and `buf2` are equal.

greater than 0 if `buf1` and `buf2` contain different bytes in the first `len` bytes, and the first such byte is larger in `buf1` than the byte at the same position in `buf2`.

Errors: None.

See also: CompareChar (1467), CompareChar0 (1469), CompareWord (1470), CompareDWord (1469)

Listing: ./examples/refex/ex99.pp

```
Program Example99;

{ Program to demonstrate the CompareByte function. }

Const
  ArraySize     = 100;
  HalfArraySize = ArraySize Div 2;

Var
  Buf1,Buf2 : Array[1..ArraySize] of byte;
  I : longint;

  Procedure CheckPos(Len : Longint);

  Begin
    Write('First ',Len,' positions are ');
    if CompareByte(Buf1,Buf2,Len)<>0 then
      Write('NOT ');
    Writeln('equal');
  end;


begin
  For I:=1 to ArraySize do
    begin
    Buf1[i]:=I;
    If I<=HalfArraySize Then
      Buf2[I]:=I
    else
      Buf2[i]:=HalfArraySize-I;
    end;
  CheckPos(HalfArraySize div 2);
  CheckPos(HalfArraySize);
  CheckPos(HalfArraySize+1);
  CheckPos(HalfArraySize + HalfArraySize Div 2);
end.
```

75.12.69 CompareChar

Synopsis: compare 2 memory buffers character per character.

Declaration: `function CompareChar(const buf1; const buf2; len: SizeInt) : SizeInt`

Visibility: default

Description: `CompareChar` compares two memory regions `buf1`,`buf2` on a character-per-character basis for a total of `len` characters.

The `CompareChar0` variant compares `len` bytes, or until a zero character is found.

The function returns one of the following values:

-1if `buf1` and `buf2` contain different characters in the first `len` positions, and the first such character is smaller in `buf1` than the character at the same position in `buf2`.

0if the first `len` characters in `buf1` and `buf2` are equal.

1if `buf1` and `buf2` contain different characters in the first `len` positions, and the first such character is larger in `buf1` than the character at the same position in `buf2`.

Errors: None.

See also: CompareByte (1466), CompareChar0 (1469), CompareWord (1470), CompareDWord (1469)

Listing: ./examples/refex/ex100.pp

```
Program Example100;

{ Program to demonstrate the CompareChar function. }

Const
  ArraySize     = 100;
  HalfArraySize = ArraySize Div 2;

Var
  Buf1,Buf2 : Array[1..ArraySize] of char;
  I : longint;

  Procedure CheckPos(Len : Longint);

  Begin
    Write('First ',Len,' characters are ');
    if CompareChar(Buf1,Buf2,Len)<>0 then
      Write('NOT ');
    Writeln('equal');
  end;

  Procedure CheckNullPos(Len : Longint);

  Begin
    Write('First ',Len,' non-null characters are ');
    if CompareChar0(Buf1,Buf2,Len)<>0 then
      Write('NOT ');
    Writeln('equal');
  end;

begin
  For I:=1 to ArraySize do
    begin
    Buf1[i]:=chr(I);
    If I<=HalfArraySize Then
      Buf2[I]:=chr(I)
    else
      Buf2[i]:=chr(HalfArraySize-I);
    end;
  CheckPos(HalfArraySize div 2);
  CheckPos(HalfArraySize);
  CheckPos(HalfArraySize+1);
  CheckPos(HalfArraySize + HalfArraySize Div 2);
  For I:=1 to 4 do
    begin
    buf1[Random(ArraySize)+1]:=Chr(0);
```

```
    buf2[Random(ArraySize)+1]:=Chr(0);
    end;
  Randomize;
  CheckNullPos(HalfArraySize div 2);
  CheckNullPos(HalfArraySize);
  CheckNullPos(HalfArraySize+1);
  CheckNullPos(HalfArraySize + HalfArraySize Div 2);
end.
```

75.12.70 CompareChar0

Synopsis: Compare two buffers character by character till a null-character is reached.

Declaration: `function CompareChar0(const buf1; const buf2; len: SizeInt) : SizeInt`

Visibility: default

Description: `CompareChar0` compares 2 buffers `buf1` and `buf2` for a maximum length of `len` or till a null character is reached in either buffer. The result depends on the contents of the buffers:

< 0 If `buf1` contains a character less than the corresponding character in `buf2`.

0 If both buffers are equal

> 0 If `buf1` contains a character greater than the corresponding character in `buf2`.

Errors: None.

See also: CompareByte (1466), CompareChar (1467), CompareDWord (1469), CompareWord (1470)

75.12.71 CompareDWord

Synopsis: Compare 2 memory buffers DWord per DWord.

Declaration: `function CompareDWord(const buf1; const buf2; len: SizeInt) : SizeInt`

Visibility: default

Description: `CompareDWord` compares two memory regions `buf1`,`buf2` on a DWord-per-DWord basis for a total of `len` DWords. (A DWord is 4 bytes).

The function returns one of the following values:

-1 if `buf1` and `buf2` contain different DWords in the first `len` DWords, and the first such DWord is smaller in `buf1` than the DWord at the same position in `buf2`.

0 if the first `len` DWords in `buf1` and `buf2` are equal.

1 if `buf1` and `buf2` contain different DWords in the first `len` DWords, and the first such DWord is larger in `buf1` than the DWord at the same position in `buf2`.

Errors: None.

See also: CompareChar (1467), CompareByte (1466), CompareWord (1470)

Listing: ./examples/refex/ex101.pp

```
Program Example101;

{ Program to demonstrate the CompareDWord function. }

Const
  ArraySize     = 100;
  HalfArraySize = ArraySize Div 2;

Var
  Buf1,Buf2 : Array[1..ArraySize] of Dword;
  I : longint;

  Procedure CheckPos(Len : Longint);

  Begin
    Write('First ',Len,' DWords are ');
    if CompareDWord(Buf1,Buf2,Len)<>0 then
      Write('NOT ');
    WriteIn('equal');
  end;


begin
  For I:=1 to ArraySize do
    begin
    Buf1[i]:=I;
    If I<=HalfArraySize Then
      Buf2[I]:=I
    else
      Buf2[i]:=HalfArraySize-I;
    end;
  CheckPos(HalfArraySize div 2);
  CheckPos(HalfArraySize);
  CheckPos(HalfArraySize+1);
  CheckPos(HalfArraySize + HalfArraySize Div 2);
end.
```

75.12.72 CompareWord

Synopsis: Compare 2 memory buffers word per word.

Declaration: `function CompareWord(const buf1; const buf2; len: SizeInt) : SizeInt`

Visibility: default

Description: `CompareWord` compares two memory regions `buf1`,`buf2` on a Word-per-Word basis for a total of `len` Words. (A Word is 2 bytes).

The function returns one of the following values:

-1 if `buf1` and `buf2` contain different Words in the first `len` Words, and the first such Word is smaller in `buf1` than the Word at the same position in `buf2`.

0 if the first `len` Words in `buf1` and `buf2` are equal.

1 if `buf1` and `buf2` contain different Words in the first `len` Words, and the first such Word is larger in `buf1` than the Word at the same position in `buf2`.

Errors: None.

See also: CompareChar (1467), CompareByte (1466), CompareDWord (1469)

Listing: ./examples/refex/ex102.pp

```
Program Example102;

{ Program to demonstrate the CompareWord function. }

Const
  ArraySize     = 100;
  HalfArraySize = ArraySize Div 2;

Var
  Buf1,Buf2 : Array[1..ArraySize] of Word;
  I : longint;

  Procedure CheckPos(Len : Longint);

  Begin
    Write('First ',Len,' words are ');
    if CompareWord(Buf1,Buf2,Len)<>0 then
      Write('NOT ');
    WriteIn('equal');
  end;


begin
  For I:=1 to ArraySize do
    begin
    Buf1[i]:=I;
    If I<=HalfArraySize Then
      Buf2[I]:=I
    else
      Buf2[i]:=HalfArraySize-I;
    end;
  CheckPos(HalfArraySize div 2);
  CheckPos(HalfArraySize);
  CheckPos(HalfArraySize+1);
  CheckPos(HalfArraySize + HalfArraySize Div 2);
end.
```

75.12.73 Concat

Synopsis: Append one string or dynamic array to another.

Declaration:
```
function Concat(const S1: string; const S2: string; const S3: string;
                const Sn: string) : string
```

Visibility: default

Description: `Concat` concatenates the strings `S1`,`S2` etc. to one long string. The same operation can be performed with the + operation.

`Concat` can also be used to concatenate 2 dynamic arrays of any type, resulting in a new dynamic array containing all the elements of the dynamic arrays used in the call.

Errors: None.

See also: Copy (1473), Delete (1477), Insert (1521), Pos (1543), Length (1528)

Listing: ./examples/refex/ex10.pp

```
Program Example10;

{ Program to demonstrate the Concat function. }
Var
  S : String;

begin
  S:=Concat('This can be done',' Easier ','with the + operator !');
end.
```

75.12.74 Continue

Synopsis: Continue with next loop cycle.

Declaration: `procedure Continue`

Visibility: default

Description: `Continue` jumps to the end of the current repetitive statement. The code between the `Continue` call and the end of the repetitive statement is skipped. The condition of the repetitive statement is then checked again.

This can be used with `For`, `repeat` and `While` statements.

Note that although `Continue` is a compiler intrinsic (i.e. is treated specially) it is defined as an identifier in the system unit, hence it can be redefined.

Errors: None.

See also: Break (1460), Exit (1488)

Listing: ./examples/refex/ex86.pp

```
Program Example86;

{ Program to demonstrate the Continue function. }

Var I : longint;

begin
  I:=0;
  While I<10 Do
    begin
    Inc(I);
    If I<5 Then
      Continue;
    WriteIn (i);
    end;
  I:=0;
  Repeat
    Inc(I);
    If I<5 Then
      Continue;
    WriteIn (i);
  Until I>=10;
```

```
  For I:=1 to 10 do
    begin
    If I<5 Then
      Continue;
    WriteLn (i);
    end;
end.
```

Listing: ./examples/refex/ex121.pp

```
{
  Example 121:
  Continue, break and exit are system procedures.
  They can be redefined
}

procedure continue;

begin
  WriteLn('Continue');
end;

Procedure Exit;

begin
  WriteLn('exit');
end;

Procedure Break;

begin
  WriteLn('Break');
end;

begin
  Repeat
    Continue;
    Break;
    exit;
  Until True;
end.
```

75.12.75 Copy

Synopsis: Copy part of a string.

Declaration:
```
function Copy(S: AStringType; Index: SizeInt; Count: SizeInt) : string
function Copy(A: DynArrayType; Index: SizeInt; Count: SizeInt)
                : DynArray
```

Visibility: default

Description: `Copy` returns a string which is a copy if the `Count` characters in `S`, starting at position `Index`. If `Count` is larger than the length of the string `S`, the result is truncated. If `Index` is larger than the length of the string `S`, then an empty string is returned. `Index` is 1-based.

For dynamic arrays, Copy returns a new dynamic array of the same type as the original one, and copies `Count` elements from the old array, starting at the position in `Index`.

The `Count` argument can be omitted. In that case, the string (or dynamic array) is copied from the position `Index` till the end of the string or array.

Errors: None.

See also: Delete (1477), Insert (1521), Pos (1543)

Listing: ./examples/refex/ex11.pp

```
Program Example11;

{ Program to demonstrate the Copy function. }

Var S,T : String;

begin
  T:='1234567';
  S:=Copy (T,1,2);    { S:='12'   }
  S:=Copy (T,4,2);    { S:='45'   }
  S:=Copy (T,4,8);    { S:='4567' }
end.
```

75.12.76 CopyArray

Synopsis: Copy managed-type elements in array.

Declaration:
```
procedure CopyArray(dest: Pointer; source: Pointer; typeInfo: Pointer;
                    count: SizeInt)
```

Visibility: default

Description: `CopyArray` copies `count` elements containing managed types from the array pointed to by `source` to the array pointed to by `dest`. For this, it uses the type information of the elements as specified in `typeinfo`.

Under normal circumstances, this procedure should not be used, it is called automatically by the compiler when an array-typed variables are assigned to each other.

See also: InitializeArray (1520), FinalizeArray (1495), DynArraySize (1482), DynArrayClear (1481), DynArrayDim (1481), DynArrayBounds (1481)

75.12.77 Cos

Synopsis: Calculate cosine of angle.

Declaration:
```
function Cos(d: ValReal) : ValReal
```

Visibility: default

Description: `Cos` returns the cosine of `X`, where X is an angle, in radians. If the absolute value of the argument is larger than $2\hat{6}3$, then the result is undefined.

Errors: None.

See also: Arctan (1450), Sin (1573)

Listing: ./examples/refex/ex12.pp

```
Program Example12;

{ Program to demonstrate the Cos function. }

Var R : Real;

begin
  R:=Cos(Pi);      { R:=-1  }
  R:=Cos(Pi/2);    { R:=0   }
  R:=Cos(0);       { R:=1   }
end.
```

75.12.78 CSeg

Synopsis: Return code segment.

Declaration: `function CSeg : Word`

Visibility: default

Description: `CSeg` returns the Code segment register. In Free Pascal, it returns always a zero, since Free Pascal is a 32/64 bit compiler.

Errors: None.

See also: DSeg (1480), Seg (1564), Ofs (1540), Ptr (1545)

Listing: ./examples/refex/ex13.pp

```
Program Example13;

{ Program to demonstrate the CSeg function. }

var W : word;

begin
  W:=CSeg; {W:=0, provided for compatibility,
                  FPC is 32 bit.}
end.
```

75.12.79 Dec

Synopsis: Decrease value of variable.

Declaration:
```
procedure Dec(var X: TOrdinal)
procedure Dec(var X: TOrdinal; Decrement: TOrdinal)
```

Visibility: default

Description: `Dec` decreases the value of `X` with `Decrement`. If `Decrement` isn't specified, then 1 is taken as a default.

`Dec` can be used on typed pointers: in that case it decreases the value with `Decrement` the size of the type the pointer points to. This works independently of the setting of the `$POINTERMATH` directive.

Errors: A range check can occur, or an underflow error, if an attempt is made to decrease `X` below its minimum value.

See also: Inc (1512)

Listing: ./examples/refex/ex14.pp

```
Program Example14;

{ Program to demonstrate the Dec function. }

Var
  I  : Integer;
  L  : Longint;
  W  : Word;
  B  : Byte;
  Si : ShortInt;

begin
 I:=1;
 L:=2;
 W:=3;
 B:=4;
 Si:=5;
 Dec (i);      { i:=0  }
 Dec (L,2);    { L:=0  }
 Dec (W,2);    { W:=1  }
 Dec (B,-2);   { B:=6  }
 Dec (Si,0);   { Si:=5 }
end.
```

75.12.80 Default

Synopsis: Return Default initialized value.

Declaration: `function Default(const T: AnyType) : AnyType`

Visibility: default

Description: `Default` is a compiler intrinsic: it returns for every type `T` a default value. In essence, this is a block of memory that is zeroed out. It can be used to correctly initialize most types (see below). Most importantly, it can be used to correctly initialize a managed type (1362). It also works using a generic type template.

This function must not be used on a type where 0 (or a block of zeroes) is not a valid value for that type. In particular, it must not be used on the file types or complex types that contain a file type. Similarly, if you use it on enumerated types and range types which do not have an element with ordinal value 0, the use of this function will result in invalid values. In both cases, the compiler will not give an error if you do use `Default` on variables of the mentioned types, but the result will be an invalid value.

See also: TypeInfo (1591), Initialize (1517), Finalize (1495), ManagedTypes (1362)

75.12.81 DefaultAnsi2UnicodeMove

Synopsis: Standard widestring manager callback.

Declaration:
```
procedure DefaultAnsi2UnicodeMove(source: PChar; cp: TSystemCodePage;
                                  var dest: unicodestring; len: SizeInt)
```

Visibility: default

Description: `DefaultAnsi2UnicodeMove` is the standard callback used for the widestring manager when an ansistring must be converted to a unicodestring. It simply copies over all characters from the ansistring to the unicodestring, no conversion whatsoever is performed.

75.12.82 DefaultAnsi2WideMove

Synopsis: Standard implementation of Ansi to Widestring conversion routine.

Declaration:
```
procedure DefaultAnsi2WideMove(source: PChar; cp: TSystemCodePage;
                               var dest: widestring; len: SizeInt)
```

Visibility: default

Description: `DefaultAnsi2WideMove` simply copies each character of the null-terminated ansi-string `Source` to the corresponding WideChar in `Dest`. At most `Len` characters will be copied.

Errors: None.

See also: DefaultUnicode2AnsiMove (1477)

75.12.83 DefaultUnicode2AnsiMove

Synopsis: Standard widestring manager callback.

Declaration:
```
procedure DefaultUnicode2AnsiMove(source: PUnicodeChar;
                                  var dest: RawByteString;
                                  cp: TSystemCodePage; len: SizeInt)
```

Visibility: default

Description: `DefaultUnicode2AnsiMove` is the standard callback used for the widestring manager when a Unicode string must be converted to an ansistring. It replaces all words with value < 256 with their value as ASCII code.

Errors: None.

See also: WidestringManager (1446)

75.12.84 Delete

Synopsis: Delete elements (characters) from a string or dynamic array.

Declaration:
```
procedure Delete(var S: string; const Index: Integer;
                 const Count: Integer)
procedure Delete(var A: DynArrayType; const Index: Integer;
                 const Count: Integer)
```

Visibility: default

Description: `Delete` removes `Count` characters from string `S`, starting at position `Index`. Index is 1-based. All characters after the deleted characters are shifted `Count` positions to the left, and the length of the string is adjusted.

For dynamic arrays, `Delete` removes `Count` elements from the array `A`, starting at position `Index`. Index is 0-based. All elements after the deleted elements are shifted `Count` positions to the left, and the length of the array is adjusted.

If the sum of `Index` and `Count` exceeds the length of the string or array, `Delete` removes the end of the string or array, starting at `Index`.

If `Index` is less than 1 or greater than the length of the string or array, or if `Count` is negative or zero, `Delete` does nothing.

See also: Copy (1473), Pos (1543), Insert (1521)

Listing: ./examples/refex/ex15.pp

```
Program Example15;

{ Program to demonstrate the Delete function. }

Var
  S : String;

begin
  S:='This is not easy !';
  Delete (S,9,4); { S:='This is easy !' }
end.
```

75.12.85 Dispose

Synopsis: Free dynamically allocated memory.

Declaration:
```
procedure &Dispose(P: Pointer)
procedure &Dispose(P: TypedPointer; Des: TProcedure)
```

Visibility: default

Description: The first form `Dispose` releases the memory allocated with a call to New (1537). The pointer `P` must be typed. The released memory is returned to the heap.

The second form of `Dispose` accepts as a first parameter a pointer to an object type, and as a second parameter the name of a destructor of this object. The destructor will be called, and the memory allocated for the object will be freed.

Errors: An runtime error will occur if the pointer doesn't point to a location in the heap.

See also: New (1537), Getmem (1502), Freemem (1499)

Listing: ./examples/refex/ex16.pp

```
Program Example16;

{ Program to demonstrate the Dispose and New functions. }

Type SS = String[20];

     AnObj = Object
```

```
        I : integer;
        Constructor Init;
        Destructor Done;
        end;

Var
  P : ^SS;
  T : ^AnObj;

Constructor Anobj.Init;

begin
 WriteIn ('Initializing an instance of AnObj !');
end;

Destructor AnObj.Done;

begin
  WriteIn ('Destroying an instance of AnObj !');
end;

begin
  New (P);
  P^:='Hello, World !';
  Dispose (P);
  { P is undefined from here on !}
  New(T,Init);
  T^.i:=0;
  Dispose (T,Done);
end.
```

75.12.86 divide(variant,variant):variant

Synopsis: Implement division (/) operation on variants.

Declaration: `operator /(const op1: variant; const op2: variant) : variant`

Visibility: default

Description: The implementation of the division / operation is delegated to the variant manager with operation `opDivide`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator *(variant, variant): variant (1362)

75.12.87 DoneCriticalSection

Synopsis: Clean up a critical section.

Declaration: `procedure DoneCriticalSection(var cs: TRTLCRITICALSECTION)`

Visibility: default

Description: `DoneCriticalsection` cleans up the critical section `CS`. After a call to `DoneCriticalsection`, the critical section can no longer be used with EnterCriticalsection (1483) or LeaveCriticalsection (1527), unless it is again initialized with InitCriticalSection (1517)

See also: InitCriticalsection (1517), EnterCriticalsection (1483), LeaveCriticalsection (1527)

75.12.88 DoneThread

Synopsis: End the current thread.

Declaration: `procedure DoneThread`

Visibility: default

Description: `DoneThread` should be used to end the current thread. It performs the necessary housekeeping before actually ending the thread. Using the operating system calls to end the thread may result in data corruption or memory leaks.

See also: BeginThread (1458)

75.12.89 DSeg

Synopsis: Return data segment.

Declaration: `function DSeg : Word`

Visibility: default

Description: `DSeg` returns the data segment register. In Free Pascal, it returns always a zero, since Free Pascal is a 32/64 bit compiler.

Errors: None.

See also: CSeg (1475), Seg (1564), Ofs (1540), Ptr (1545)

Listing: ./examples/refex/ex17.pp

```
Program Example17;

{ Program to demonstrate the DSeg function. }

Var
  W : Word;

begin
  W:=DSeg; {W:=0, This function is provided for compatibility,
                  FPC is a 32 bit compiler.}
end.
```

75.12.90 DumpExceptionBacktrace

Synopsis: Create backtrace.

Declaration: `procedure DumpExceptionBacktrace(var f: text)`

Visibility: default

Description: `DumpExceptionBackTrace` writes a backtrace of the current exception to the file `f`. If no exception is currently being raised, nothing is written. As much frames as available are written. If debug info is available, then file names and line numbers will be written as well.

Errors: No check is done to see whether `f` is opened for writing.

See also: dump_stack (1481)

75.12.91 Dump_Stack

Synopsis: Dump stack to the given text file.

Declaration:
```
procedure Dump_Stack(var f: text; fp: pointer; addr: CodePointer)
procedure Dump_Stack(var f: text; skipframes: LongInt)
```

Visibility: default

Description: `Dump_Stack` prints a stack dump to the file `f`, with base frame pointer `bp`

Errors: The file `f` must be opened for writing or an error will occur.

See also: get_caller_addr (1507), get_caller_frame (1507), get_frame (1508)

75.12.92 DynArrayBounds

Synopsis: Return the bounds of the dynamic array.

Declaration:
```
function DynArrayBounds(a: Pointer; typeInfo: Pointer) : TBoundArray
```

Visibility: default

Description: `DynArrayBounds` returns the bounds of all the dimensions of the dynamic array `a` with type information `typeInfo`.

The result is an array (zero-based) with the maximum valid index for each dimension in the array: the lower bound is not present in the result, it is always zero.

See also: InitializeArray (1520), FinalizeArray (1495), CopyArray (1474), DynArraySize (1482), DynArrayClear (1481), DynArrayDim (1481)

75.12.93 DynArrayClear

Synopsis: Clears a dynamic array.

Declaration:
```
procedure DynArrayClear(var a: Pointer; typeInfo: Pointer)
```

Visibility: default

Description: `DynArrayClear` clears the array (`a`) using its type info (`typeInfo`). It is equal to setting the length to zero.

See also: InitializeArray (1520), FinalizeArray (1495), CopyArray (1474), DynArraySize (1482), DynArrayDim (1481), DynArrayBounds (1481)

75.12.94 DynArrayDim

Synopsis: Return the number of dimensions in a dynamic array.

Declaration:
```
function DynArrayDim(typeInfo: Pointer) : Integer
```

Visibility: default

Description: `DynArrayDim` returns the number of dimensions in a dynamic array, using the type information (`typeInfo`) of the array.

See also: InitializeArray (1520), FinalizeArray (1495), CopyArray (1474), DynArraySize (1482), DynArrayClear (1481), DynArrayBounds (1481)

75.12.95 DynArrayIndex

Synopsis: Return pointer to indicated element.

Declaration:
```
function DynArrayIndex(a: Pointer; const indices: Array of SizeInt;
                       typeInfo: Pointer) : Pointer
```

Visibility: default

Description: `DynArrayIndex` returns a pointer to the element indicated by `indices` in dynamic array `a` with type information `typinfo`. The length of `indices` must equal the number of dimensions of the array (as returned by DynArrayDim (1481)).

Errors: No bounds checking is performed, it is therefor possible to get an access violation if one of the indexes is out of range.

See also: InitializeArray (1520), FinalizeArray (1495), CopyArray (1474), DynArraySize (1482), DynArrayClear (1481), DynArrayBounds (1481), DynArrayDim (1481), IsDynArrayRectangular (1526)

75.12.96 DynArraySetLength

Synopsis: Set the length of a dynamic array.

Declaration:
```
procedure DynArraySetLength(var a: Pointer; typeInfo: Pointer;
                            dimCnt: SizeInt; lengthVec: PSizeInt)
```

Visibility: default

Description: `DynArraySetLength` sets the length of the dynamic array `a` to the first `dimCnt` lengths specified in the array `lengthVec`. The dynamic array type is described in `typeInfo` which points to a record of type TDynArrayTypeInfo (1414)

It should never be necessary to call this function directly, the standard SetLength (1566) function should be used instead.

Errors: If an invalid pointer is specified, an error may occur.

See also: SetLength (1566), tdynarraytypeinfo (1414)

75.12.97 DynArraySize

Synopsis: Return length of dynamic array.

Declaration:
```
function DynArraySize(a: pointer) : tdynarrayindex
```

Visibility: default

Description: `DynArraySize` gets the number of elements in the array (`a`) the result is equal to Length (1528) for dynamic arrays.

See also: InitializeArray (1520), FinalizeArray (1495), CopyArray (1474), DynArrayClear (1481), DynArrayDim (1481), DynArrayBounds (1481)

75.12.98 EmptyMethod

Synopsis: Empty method alias.

Declaration:
```
procedure EmptyMethod
```

Visibility: default

Description: `Emptymethod` is meant for the compiler only. It should not be used directly.

75.12.99 EndThread

Synopsis: End the current thread.

Declaration:
```
procedure EndThread(ExitCode: DWord)
procedure EndThread
```

Visibility: default

Description: `EndThread` ends the current thread. If `ExitCode` is supplied, it is returned as the exit code for the thread to a function waiting for the thread to terminate (WaitForThreadTerminate (1600)). If it is omitted, zero is used.

This function does not return.

See also: WaitForThreadTerminate (1600), BeginThread (1458)

75.12.100 EnterCriticalSection

Synopsis: Enter a critical section.

Declaration:
```
procedure EnterCriticalSection(var cs: TRTLCRITICALSECTION)
```

Visibility: default

Description: `EnterCriticalSection` will suspend the current thread if another thread has currently entered the critical section. When the other thread has left the critical section (through LeaveCriticalSection (1527)), the current thread resumes execution. The result is that only 1 thread is executing code which is protected by a `EnterCriticalsection` and `LeaveCriticalSection` pair.

The critical section must have been initialized with InitCriticalSection (1517) prior to a call to `EnterCriticalsection`.

A call to `EnterCriticalsection` must always be matched by a call to LeaveCriticalSection (1527). To avoid problems, it is best to include the code to be execute in a `try...finally` block, as follows:

```
EnterCriticalSection(Section);
  Try
    // Code to be protected goes here.
  Finally
    LeaveCriticalSection(Section);
  end;
```

For performance reasons it is best to limit the code between the entering and leaving of a critical section as short as possible.

See also: InitCriticalsection (1517), DoneCriticalsection (1479), LeaveCriticalsection (1527)

75.12.101 EnumResourceLanguages

Synopsis: Enumerate available languages for a resource of given type and name.

Declaration:
```
function EnumResourceLanguages(ModuleHandle: TFPResourceHMODULE;
                               ResourceType: PChar; ResourceName: PChar;
                               EnumFunc: EnumResLangProc; lParam: PtrInt)
                                : LongBool
```

Visibility: default

Description: `EnumResourceLanguages` enumerates the available languages for a resource of given `ResourceName` and type `ResourceType` in the module `ModuleHandle`. For each language available, it calls `EnumFunc` and passes it `ModuleHandle`, the type of the resource `ResourceType`, the name of the resource `ResourceName`, the language ID, and `lParam`. It returns `False` if no resources are available for the specified resource type and module, or `True` if there are resources available.

Errors: None.

See also: EnumResourceTypes (1484), EnumResourceNames (1484), EnumResourceLanguages (1483)

75.12.102 EnumResourceNames

Synopsis: Enumerate available resource names for a specified resource type.

Declaration:
```
function EnumResourceNames(ModuleHandle: TFPResourceHMODULE;
                           ResourceType: PChar;
                           EnumFunc: EnumResNameProc; lParam: PtrInt)
                            : LongBool
```

Visibility: default

Description: `EnumResourceNames` enumerates the names of all resources of type `ResourceType` in the module `ModuleHandle`. For each resource available it calls `EnumFunc` and passes it `ModuleHandle`, the type of the resource `ResourceType`, the name of the resource, and `lParam`. It returns `False` if no resources are available for the specified resource type and module, or `True` if there are resources available.

Errors: None.

See also: EnumResourceTypes (1484), EnumResourceLanguages (1483)

75.12.103 EnumResourceTypes

Synopsis: Enumerate available resource types.

Declaration:
```
function EnumResourceTypes(ModuleHandle: TFPResourceHMODULE;
                           EnumFunc: EnumResTypeProc; lParam: PtrInt)
                            : LongBool
```

Visibility: default

Description: `EnumResourceTypes` enumerates the types of all resources in the module `ModuleHandle`. For each resource available it calls `EnumFunc` and passes it `ModuleHandle`, the type of the resource, and `lParam`. It returns `False` if no resources are available for the specified module, or `True` if there are resources available.

Errors: None.

See also: EnumResourceNames (1484), EnumResourceLanguages (1483)

75.12.104 EOF

Synopsis: Check for end of file.

Declaration:
```
function EOF(var f: File) : Boolean
function EOF(var t: Text) : Boolean
function EOF : Boolean
```

Visibility: default

Description: `Eof` returns `True` if the file-pointer has reached the end of the file, or if the file is empty. In all other cases `Eof` returns `False`. If no file `F` is specified, standard input is assumed.

Note that calling this function may cause your program to wait: to determine whether you are at EOF, it is necessary to read data. If the file descriptor is not a real file (for instance for standard input or sockets), then this call may seem to hang the program while it is waiting for data to appear or for the file descriptor to be closed.

Errors: Depending on the state of the `{$I}` switch, a runtime error can be generated if there is an error. In the `{$I-}` state, use `IOResult` to check for errors.

See also: Eoln (1485), Assign (1451), Reset (1552), Rewrite (1553)

Listing: ./examples/refex/ex18.pp

```
Program Example18;

{ Program to demonstrate the Eof function. }

Var T1,T2 : text;
    C : Char;

begin
  { Set file to read from. Empty means from standard input.}
  assign (t1,paramstr(1));
  reset (t1);
  { Set file to write to. Empty means to standard output. }
  assign (t2,paramstr(2));
  rewrite (t2);
  While not eof(t1) do
    begin
    read (t1,C);
    write (t2,C);
    end;
  Close (t1);
  Close (t2);
end.
```

75.12.105 EOLn

Synopsis: Check for end of line.

Declaration:
```
function EOLn(var t: Text) : Boolean
function EOLn : Boolean
```

Visibility: default

Description: `EoLn` returns `True` if the file pointer has reached the end of a line, which is demarcated by a line-feed character (ASCII value 10), or if the end of the file is reached. In all other cases `EoLn` returns `False`. If no file `F` is specified, standard input is assumed. It can only be used on files of type `Text`.

Errors: None.

See also: Eof (1484), Assign (1451), Reset (1552), Rewrite (1553)

Listing: ./examples/refex/ex19.pp

```
Program Example19;

{ Program to demonstrate the Eoln function. }

begin
  { This program waits for keyboard input. }
  { It will print True when an empty line is put in,
    and false when you type a non-empty line.
    It will only stop when you press enter.}
  While not Eoln do
    WriteLn (eoln);
end.
```

75.12.106 equal(variant,variant):Boolean

Synopsis: Implement = (equality) operation on variants.

Declaration: `operator =(const op1: variant; const op2: variant) : Boolean`

Visibility: default

Description: The implementation of the equality `(=)` operation is delegated to the variant manager with operation `opcmpeq`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator <(variant, variant): boolean (1362)

75.12.107 Erase

Synopsis: Delete a file from disk.

Declaration:
```
procedure Erase(var f: File)
procedure Erase(var t: Text)
```

Visibility: default

Description: `Erase` removes an unopened file from disk. The file should be assigned with `Assign`, but not opened with `Reset` or `Rewrite`

Errors: Depending on the state of the `{$I}` switch, a runtime error can be generated if there is an error. In the `{$I-}` state, use `IOResult` to check for errors.

See also: Assign (1451)

Listing: ./examples/refex/ex20.pp

```
Program Example20;

{ Program to demonstrate the Erase function. }

Var F : Text;

begin
  { Create a file with a line of text in it}
  Assign (F,'test.txt');
```

```
  Rewrite (F);
  WriteLn (F,'Try and find this when I''m finished !');
  close (f);
  { Now remove the file }
  Erase (f);
end.
```

75.12.108 Error

Synopsis: Generate run-time error.

Declaration: `procedure Error(RunTimeError: TRuntimeError)`

Visibility: default

Description: `Error` generates a run-time error with an exit code corresponding to `RunTimeError`. This function is implemented for Delphi compatibility, and is not used by the Free Pascal Run-Time Library.

See also: RunError (1560), Halt (1509)

75.12.109 Exclude

Synopsis: Exclude element from a set if it is present.

Declaration: `procedure Exclude(var S: TSetType; E: TSetElement)`

Visibility: default

Description: `Exclude` removes `E` from the set `S` if it is included in the set. E should be of the same type as the base type of the set `S`.

Thus, the two following statements do the same thing:

```
S:=S-[E];
Exclude(S,E);
```

Errors: If the type of the element `E` is not equal to the base type of the set `S`, the compiler will generate an error.

See also: Include (1513)

Listing: ./examples/refex/ex111.pp

```
program Example111;

{ Program to demonstrate the Include/Exclude functions }

Type
  TEnumA = (aOne,aTwo,aThree);
  TEnumAs = Set of TEnumA;

Var
  SA : TEnumAs;

  Procedure PrintSet(S : TEnumAs);

  var
```

```
    B : Boolean;

    procedure DoEl(A : TEnumA; Desc : String);

    begin
      If A in S then
        begin
        If B then
          Write(',');
        B:=True;
        Write(Desc);
        end;
    end;

  begin
    Write('[');
    B:=False;
    DoEl(aOne,'aOne');
    DoEl(aTwo,'aTwo');
    DoEl(aThree,'aThree');
    WriteIn(']')
  end;

begin
  SA:=[];
  Include(SA,aOne);
  PrintSet(SA);
  Include(SA,aThree);
  PrintSet(SA);
  Exclude(SA,aOne);
  PrintSet(SA);
  Exclude(SA,aTwo);
  PrintSet(SA);
  Exclude(SA,aThree);
  PrintSet(SA);
end.
```

75.12.110 Exit

Synopsis: Exit current subroutine.

Declaration:
```
procedure &Exit(const X: TAnyType)
procedure &Exit
```

Visibility: default

Description: `Exit` exits the current subroutine, and returns control to the calling routine. If invoked in the main program routine, exit stops the program. The optional argument `X` allows to specify a return value, in the case `Exit` is invoked in a function. The function result will then be equal to `X`.

In Object Pascal or Delphi modes, if the `Exit` statement is surrounded by one or more `Try .. Finally` constructs, the `Finally` blocks are executed, which means that if the finally blocks are used to free resources, then these resources will also be freed when `Exit` is called.

Note that although `Exit` is a compiler intrinsic (i.e. is treated specially) it is defined as an identifier in the system unit, hence it can be redefined.

Errors: None.

See also: Halt (1509)

Listing: ./examples/refex/ex21.pp

```
Program Example21;

{ Program to demonstrate the Exit function. }

Procedure DoAnExit (Yes : Boolean);

{ This procedure demonstrates the normal Exit }

begin
  Writeln ('Hello from DoAnExit !');
  If Yes then
    begin
    Writeln ('Bailing out early.');
    exit;
    end;
  Writeln ('Continuing to the end.');
end;

Function Positive (Which : Integer) : Boolean;

{ This function demonstrates the extra FPC feature of Exit :
  You can specify a return value for the function }

begin
  if Which>0 then
    exit (True)
  else
    exit (False);
end;

begin
  { This call will go to the end }
  DoAnExit (False);
  { This call will bail out early }
  DoAnExit (True);
  if Positive (-1) then
    Writeln ('The compiler is nuts, -1 is not positive.')
  else
    Writeln ('The compiler is not so bad, -1 seems to be negative.');
end.
```

Listing: ./examples/refex/ex121.pp

```
{
  Example 121:
  Continue, break and exit are system procedures.
  They can be redefined
}

procedure continue;

begin
  Writeln('Continue');
end;
```

```
Procedure Exit;

begin
  Writeln('exit');
end;

Procedure Break;

begin
  Writeln('Break');
end;

begin
  Repeat
    Continue;
    Break;
    exit;
  Until True;
end.
```

75.12.111 Exp

Synopsis: Exponentiate.

Declaration: `function Exp(d: ValReal) : ValReal`

Visibility: default

Description: `Exp` returns the exponent of `X`, i.e. the number `e` to the power `X`.

Errors: None.

See also: Ln (1530)

Listing: ./examples/refex/ex22.pp

```
Program Example22;

{ Program to demonstrate the Exp function. }

begin
  Writeln (Exp(1):8:2); { Should print 2.72 }
end.
```

75.12.112 Fail

Synopsis: Fail a constructor.

Declaration: `procedure Fail`

Visibility: default

Description: `Fail` can be used in a constructor for an object or class. It will exit the constructor at once, and the memory allocated for the constructor is freed. This mean that for objects allocated with New (1537), the resulting pointer is `Nil` and for classes, the object instance will be `Nil`.

See also: TypeOf (1591), New (1537), Initialize (1517), Finalize (1495)

Listing: ./examples/refex/ex116.pp

```
program testfail;

{$mode objfpc}

Type
  TMyClass = Class
    Constructor Create;
  end;


Constructor TMyClass.Create;

begin
  Fail;
end;

var
  M : TMyClass;

begin
  M:=TMyClass.Create;
  Writeln('M is nil : ',Not Assigned(M));
end.
```

75.12.113 FilePos

Synopsis: Get position in file.

Declaration: `function FilePos(var f: File) : Int64`

Visibility: default

Description: `Filepos` returns the current record position of the file-pointer in file `F`. It cannot be invoked with a file of type `Text`. A compiler error will be generated if this is attempted. Untyped files have a default record size of 128, if the second parameter to Reset (1552) isn't specified.

Errors: Depending on the state of the `{$I}` switch, a runtime error can be generated if there is an error. In the `{$I-}` state, use `IOResult` to check for errors.

See also: Filesize (1492)

Listing: ./examples/refex/ex23.pp

```
Program Example23;

{ Program to demonstrate the FilePos function. }

Var F : File of Longint;
    L,FP : longint;

begin
  { Fill a file with data :
    Each position contains the position ! }
  Assign (F,'test.tmp');
  Rewrite (F);
  For L:=0 to 100 do
```

```
    begin
    FP:=FilePos(F);
    Write (F,FP);
    end;
  Close (F);
  Reset (F);
  { If all goes well, nothing is displayed here. }
  While not (Eof(F)) do
    begin
    FP:=FilePos (F);
    Read (F,L);
    if L<>FP then
      Writeln ('Something wrong: Got ',I,' on pos ',FP);
    end;
  Close (F);
  Erase (f);
end.
```

75.12.114 FileSize

Synopsis: Size of file.

Declaration: `function FileSize(var f: File) : Int64`

Visibility: default

Description: `Filesize` returns the total number of records in file `F`. It cannot be invoked with a file of type `Text`. (under Linux and Unix, this also means that it cannot be invoked on pipes). If `F` is empty, 0 is returned. Untyped files have a default record size of 128, if the second parameter to Reset (1552) isn't specified.

Note that the file must be open for this function to return a result.

Errors: Depending on the state of the `{$I}` switch, a runtime error can be generated if there is an error. In the `{$I-}` state, use `IOResult` to check for errors.

See also: Filepos (1491)

Listing: ./examples/refex/ex24.pp

```
Program Example24;

{ Program to demonstrate the FileSize function. }

Var F : File Of byte;
    L : File Of Longint;

begin
  Assign (F,paramstr(1));
  Reset (F);
  Writeln ('File size in bytes : ',FileSize(F));
  Close (F);
  Assign (L,paramstr (1));
  Reset (L);
  Writeln ('File size in Longints : ',FileSize(L));
  Close (f);
end.
```

75.12.115 FillByte

Synopsis: Fill memory region with 8-bit pattern.

Declaration:
```
procedure FillByte(var x; count: SizeInt; value: Byte)
```

Visibility: default

Description: `FillByte` fills the memory starting at `X` with `Count` bytes with value equal to `Value`. This is useful for quickly zeroing out a memory location. When the size of the memory location to be filled out is a multiple of 2 bytes, it is better to use Fillword (1494), and if it is a multiple of 4 bytes it is better to use FillDWord (1494), these routines are optimized for their respective sizes.

Errors: No checking on the size of `X` is done.

See also: Fillchar (1493), FillDWord (1494), Fillword (1494), Move (1536)

Listing: ./examples/refex/ex103.pp

```
Program Example103;

{ Program to demonstrate the FillByte function. }

Var S : String[10];
    I : Byte;

begin
  For i:=10 downto 0 do
    begin
    { Fill S with i bytes }
    FillByte (S,SizeOf(S),32);
    { Set Length }
    SetLength(S,I);
    WriteIn (s,'*');
    end;
end.
```

75.12.116 FillChar

Synopsis: Fill memory region with certain character.

Declaration:
```
procedure FillChar(var x; count: SizeInt; Value: Byte)
procedure FillChar(var x; count: SizeInt; Value: Boolean)
procedure FillChar(var x; count: SizeInt; Value: Char)
```

Visibility: default

Description: `Fillchar` fills the memory starting at `X` with `Count` bytes or characters with value equal to `Value`.

Errors: No checking on the size of `X` is done.

See also: Fillword (1494), Move (1536), FillByte (1493), FillDWord (1494)

Listing: ./examples/refex/ex25.pp

```
Program Example25;

{ Program to demonstrate the FillChar function. }

Var S : String[10];
    I : Byte;
begin
  For i:=10 downto 0 do
    begin
    { Fill S with i spaces }
    FillChar (S,SizeOf(S),' ');
    { Set Length }
    SetLength(S,I);
    WriteLn (s,'*');
    end;
end.
```

75.12.117 FillDWord

Synopsis: Fill memory region with 32-bit pattern.

Declaration: `procedure FillDWord(var x; count: SizeInt; value: DWord)`

Visibility: default

Description: `Fillword` fills the memory starting at `X` with `Count` DWords with value equal to `Value`. A DWord is 4 bytes in size.

Errors: No checking on the size of `X` is done.

See also: FillByte (1493), Fillchar (1493), Fillword (1494), Move (1536)

Listing: ./examples/refex/ex104.pp

```
Program Example104;

{ Program to demonstrate the FillDWord function. }

Const
  ArraySize = 1000;

Var
  S : Array [1..ArraySize] of DWord;
  I : longint;

begin
  FillDWord(S,ArraySize,0);
  For I:=1 to ArraySize do
    If S[i]<>0 then
      WriteIn('Position ',i,' not zeroed out');
end.
```

75.12.118 FillWord

Synopsis: Fill memory region with 16-bit pattern.

Declaration: `procedure FillWord(var x; count: SizeInt; Value: Word)`

Visibility: default

Description: `Fillword` fills the memory starting at `X` with `Count` words with value equal to `Value`. A word is 2 bytes in size.

Errors: No checking on the size of `X` is done.

See also: Fillchar (1493), Move (1536)

Listing: ./examples/refex/ex76.pp

```
Program Example76;

{ Program to demonstrate the FillWord function. }

Var W : Array[1..100] of Word;

begin
  { Quick initialization of array W }
  FillWord(W,100,0);
end.
```

75.12.119 Finalize

Synopsis: Finalize (clean up) memory block using RTTI.

Declaration: `procedure Finalize(var T: TAnyType; ACount: SizeInt)`

Visibility: default

Description: `Finalize` is a compiler intrinsic: it cleans up (finalizes) a memory area belonging to a variable `T` of a managed type (1362) (`TManagedType`). Finalizing means decreasing reference counts where necessary and generally zeroing out the memory area. It performs the opposite operation of initialize (1517).

The optional `ACount` parameter can be used to finalize an array of values.

For examples, see Initialize (1517).

See also: Initialize (1517), Default (1476), TypeInfo (1591), ManagedTypes (1362)

75.12.120 FinalizeArray

Synopsis: Finalize managed-type elements in array.

Declaration: `procedure FinalizeArray(p: Pointer; typeInfo: Pointer; count: SizeInt)`

Visibility: default

Description: `FinalizeArray` dereferences and clears managed types in the array pointed to by `p`. For this, it uses the type information of the elements as specified in `typeinfo`.

Under normal circumstances, this procedure should not be used, it is called automatically by the compiler when an array-typed variable containing managed types goes out of scope.

See also: InitializeArray (1520), CopyArray (1474), DynArraySize (1482), DynArrayClear (1481), DynArrayDim (1481), DynArrayBounds (1481)

75.12.121 FindResource

Synopsis: Locate a resource and return a handle to it.

Declaration:
```
function FindResource(ModuleHandle: TFPResourceHMODULE;
                      ResourceName: PChar; ResourceType: PChar)
                       : TFPResourceHandle
function FindResource(ModuleHandle: TFPResourceHMODULE;
                      const ResourceName: AnsiString;
                      const ResourceType: AnsiString) : TFPResourceHandle
function FindResource(ModuleHandle: TFPResourceHMODULE;
                      const ResourceName: AnsiString;
                      ResourceType: PChar) : TFPResourceHandle
function FindResource(ModuleHandle: TFPResourceHMODULE;
                      ResourceName: PChar;
                      const ResourceType: AnsiString) : TFPResourceHandle
```

Visibility: default

Description: `FindResource` searches for a resource with name `ResourceName` and of type `ResourceType` in the executable or library identified by `ModuleHandle`. It returns a `TResourceHandle` which can be used to load the resource with LoadResource (1531).

Errors: None. In case the resource was not found, 0 is returned.

See also: FreeResource (1500), LoadResource (1531), SizeofResource (1574), LockResource (1532), UnlockResource (1594), FreeResource (1500)

75.12.122 FindResourceEx

Synopsis: Find a resource based on type, name, language.

Declaration:
```
function FindResourceEx(ModuleHandle: TFPResourceHMODULE;
                        ResourceType: PChar; ResourceName: PChar;
                        Language: Word) : TFPResourceHandle
function FindResourceEx(ModuleHandle: TFPResourceHMODULE;
                        const ResourceType: AnsiString;
                        const ResourceName: AnsiString; Language: Word)
                         : TFPResourceHandle
function FindResourceEx(ModuleHandle: TFPResourceHMODULE;
                        ResourceType: PChar;
                        const ResourceName: AnsiString; Language: Word)
                         : TFPResourceHandle
function FindResourceEx(ModuleHandle: TFPResourceHMODULE;
                        const ResourceType: AnsiString;
                        ResourceName: PChar; Language: Word)
                         : TFPResourceHandle
```

Visibility: default

Description: `FindResourceEx` looks in module `ModuleHandle` for a resource of type `ResourceType` and name `ResourceName` with language ID `Language`. Both `Resourcename` and `ResourceName` can be specified as a null-terminated array of characters, or as an AnsiString.

If the requested language/sublanguage is not found, then the search is conducted

1.with only primary language.

2.with the neutral language (LANG_NEUTRAL)

3.with the English language

If none of these has returned a match, then the first available language is returned.

If a match is found, a handle to the resource is returned. If none is found, an empty handle (nil or 0) is returned.

Errors: None.

75.12.123 float_raise

Synopsis: Raise floating point exception.

Declaration:
```
procedure float_raise(i: TFPUException)
procedure float_raise(i: TFPUExceptionMask)
```

Visibility: default

Description: float_raise raises the floating point exceptions specified by softfloat_exception_flags (1445).

See also: softfloat_exception_flags (1445), softfloat_exception_mask (1445)

75.12.124 Flush

Synopsis: Write file buffers to disk.

Declaration:
```
procedure Flush(var t: Text)
```

Visibility: default

Description: Flush empties the internal buffer of an opened file F and writes the contents to disk. The file is **not** closed as a result of this call.

Errors: Depending on the state of the {$I} switch, a runtime error can be generated if there is an error. In the {$I-} state, use IOResult to check for errors.

See also: Close (1465)

Listing: ./examples/refex/ex26.pp

```
Program Example26;

{ Program to demonstrate the Flush function. }

Var F : Text;

begin
  { Assign F to standard output }
  Assign (F,'');
  Rewrite (F);
  Writeln (F,'This line is written first, but appears later !');
  { At this point the text is in the internal pascal buffer,
    and not yet written to standard output }
  Writeln ('This line appears first, but is written later !');
  { A writeln to 'output' always causes a flush - so this text is
    written to screen }
  Flush (f);
```

```
  { At this point, the text written to F is written to screen. }
  Write (F,'Finishing ');
  Close (f); { Closing a file always causes a flush first }
  WriteLn ('off.');
end.
```

75.12.125 FlushThread

Synopsis: Flush all standard files.

Declaration: `procedure FlushThread`

Visibility: default

Description: `FlushThread` flushes any buffers from standard file descriptors such as standard input/output/error. It should normally not be called by user code, but is executed when a thread exits.

See also: EndThread (1483)

75.12.126 FMADouble

Synopsis: Internal function, do not use.

Declaration: `function FMADouble(d1: Double; d2: Double; d3: Double) : Double`

Visibility: default

75.12.127 FMAExtended

Synopsis: Internal function, do not use.

Declaration:
```
function FMAExtended(e1: extended; e2: extended; e3: extended)
                          : extended
```

Visibility: default

75.12.128 FMASingle

Synopsis: Internal function, do not use.

Declaration: `function FMASingle(s1: single; s2: single; s3: single) : single`

Visibility: default

75.12.129 FPower10

Synopsis: Fast multiply with a power of 10.

Declaration: `function FPower10(val: Extended; Power: LongInt) : Extended`

Visibility: default

Description: `FPower10` multplies `val` with 10 to the power `Power`. It uses a fast algorithm to calculate the result.

75.12.130 Frac

Synopsis: Return fractional part of floating point value.

Declaration:
```
function Frac(d: ValReal) : ValReal
```

Visibility: default

Description: `Frac` returns the non-integer part of `X`.

Errors: None.

See also: Round (1557), Int (1521)

Listing: ./examples/refex/ex27.pp

```
Program Example27;

{ Program to demonstrate the Frac function. }

Var R : Real;

begin
  WriteLn (Frac (123.456):0:3);  { Prints  0.456 }
  WriteLn (Frac (-123.456):0:3); { Prints -0.456 }
end.
```

75.12.131 FreeLibrary

Synopsis: For compatibility with Delphi/Windows: Unload a library.

Declaration:
```
function FreeLibrary(Lib: TLibHandle) : Boolean
```

Visibility: default

Description: `FreeLibrary` provides the same functionality as UnloadLibrary (1594), and is provided for compatibility with Delphi.

See also: UnloadLibrary (1594)

75.12.132 Freemem

Synopsis: Release allocated memory.

Declaration:
```
procedure Freemem(p: pointer; Size: PtrUInt)
function Freemem(p: pointer) : PtrUInt
```

Visibility: default

Description: `Freemem` releases the memory occupied by the pointer `P`, of size `Count` (in bytes), and returns it to the heap. `P` should point to the memory allocated to a dynamic variable.

Errors: An error will occur when `P` doesn't point to the heap.

See also: Getmem (1502), New (1537), Dispose (1478)

Listing: ./examples/refex/ex28.pp

```
Program Example28;

{ Program to demonstrate the FreeMem and GetMem functions. }

Var P : Pointer;
    MM : Longint;

begin
  { Get memory for P }
  GetMem (P,80);
  FillChar (P^,80,' ');
  FreeMem (P,80);
end.
```

75.12.133 Freememory

Synopsis: Alias for FreeMem (1499).

Declaration:
```
procedure Freememory(p: pointer; Size: PtrUInt)
function Freememory(p: pointer) : PtrUInt
```

Visibility: default

Description: `FreeMemory` is an alias for FreeMem (1499).

See also: FreeMem (1499)

75.12.134 FreeResource

Synopsis: Free a loaded resource.

Declaration:
```
function FreeResource(ResData: TFPResourceHGLOBAL) : LongBool
```

Visibility: default

Description: `FreeResource` unloads the resource identified by `ResData` from memory. The resource must have been loaded by LoadResource (1531). It returns `True` if the operation was successful, `False` otherwise.

Errors: On error, `False` is returned.

See also: FindResource (1496), LoadResource (1531), SizeofResource (1574), LockResource (1532), UnlockResource (1594), FreeResource (1500)

75.12.135 Get8087CW

Declaration:
```
function Get8087CW : Word
```

Visibility: default

75.12.136 GetCPUCount

Synopsis: Return the number of cores on the system.

Declaration:
```
function GetCPUCount : LongWord
```

Visibility: default

Description: `GetCPUCount` returns the number of CPU cores on the system. Whether these are physically separate CPUs or cores on a single CPU is deliberately undefined.

See also: CPUCount (1362)

75.12.137 GetCurrentThreadId

Synopsis: Return the id of the currently running thread.

Declaration: `function GetCurrentThreadId : TThreadID`

Visibility: default

Description: `GetCurrentThreadId` returns the ID of the currently running thread. It can be used in calls such as KillThread (1527) or ThreadSetPriority (1588)

Errors: None.

See also: KillThread (1527), ThreadSetPriority (1588)

75.12.138 GetDir

Synopsis: Return the current directory.

Declaration:
```
procedure GetDir(drivenr: Byte; var dir: shortstring);  Overload
procedure GetDir(drivenr: Byte; var dir: RawByteString);  Overload
procedure GetDir(drivenr: Byte; var dir: unicodestring);  Overload
```

Visibility: default

Description: `Getdir` returns in `dir` the current directory on the drive `drivenr`, where {drivenr} is 1 for the first floppy drive, 3 for the first hard disk etc. A value of 0 returns the directory on the current disk. On Linux and Unix systems, `drivenr` is ignored, as there is only one directory tree.

Errors: An error is returned under dos, if the drive requested isn't ready.

See also: Chdir (1464)

Listing: ./examples/refex/ex29.pp

```
Program Example29;

{ Program to demonstrate the GetDir function. }

Var S : String;

begin
  GetDir (0,S);
  Writeln ('Current directory is : ',S);
end.
```

75.12.139 GetDynLibsManager

Synopsis: Return currently active dynamic library support handler.

Declaration:
```
procedure GetDynLibsManager(var Manager: TDynLibsManager)
```

Visibility: default

Description: `GetDynLibsManager` returns the currently active dynamic library support handler. This handler has normally been set by inclusion of the #rtl.dynlibs (734) unit.

See also: #rtl.dynlibs (734), SetDynLibsManager (1565)

75.12.140 GetFPCHeapStatus

Synopsis: Return FPC heap manager status information.

Declaration:
```
function GetFPCHeapStatus : TFPCHeapStatus
```

Visibility: default

Description: Return FPC heap manager status information.

75.12.141 GetHeapStatus

Synopsis: Return the memory manager heap status.

Declaration:
```
function GetHeapStatus : THeapStatus
```

Visibility: default

75.12.142 GetLoadErrorStr

Synopsis: Return an error describing the last library loading error.

Declaration:
```
function GetLoadErrorStr : string
```

Visibility: default

Description: `GetLoadErrorStr` returns an error string describing the last library loading error. This function must be called before any other OS calls are performed.

Errors: None.

See also: LoadLibrary (1531), SafeLoadLibrary (1561)

75.12.143 GetMem

Synopsis: Allocate new memory on the heap.

Declaration:
```
procedure Getmem(out p: pointer; Size: PtrUInt)
function GetMem(size: PtrUInt) : pointer
```

Visibility: default

Description: `Getmem` reserves `Size` bytes memory on the heap, and returns a pointer to this memory in `p`. What happens if no more memory is available, depends on the value of the variable ReturnNilIfGrowHeapfails (1445): if the variable is `True` then `Nil` is returned. If the variable is `False`, a run-time error is generated. The default value is `False`, so by default an error is generated.

The newly allocated memory is not initialized in any way, and may contain garbage data. It must be cleared with a call to FillChar (1493) or FillWord (1494).

For an example, see Freemem (1499).

Errors: None.

See also: Freemem (1499), Dispose (1478), New (1537), returnnilifgrowheapfails (1445), MemSize (1535)

75.12.144 GetMemory

Synopsis: Alias for GetMem (1502).

Declaration:
```
procedure Getmemory(out p: pointer; Size: PtrUInt)
function GetMemory(size: PtrUInt) : pointer
```

Visibility: default

Description: `Getmemory` is an alias for GetMem (1502).

See also: GetMem (1502)

75.12.145 GetMemoryManager

Synopsis: Return current memory manager.

Declaration:
```
procedure GetMemoryManager(var MemMgr: TMemoryManager)
```

Visibility: default

Description: `GetMemoryManager` stores the current Memory Manager record in `MemMgr`.

For an example, see the programmer's guide.

Errors: None.

See also: SetMemoryManager (1567), IsMemoryManagerSet (1527)

75.12.146 GetMXCSR

Declaration:
```
function GetMXCSR : DWord
```

Visibility: default

75.12.147 GetProcAddress

Synopsis: For compatibility with Delphi/Windows: Get the address of a procedure.

Declaration:
```
function GetProcAddress(Lib: TLibHandle; const ProcName: AnsiString)
                        : Pointer
```

Visibility: default

Description: `GetProcAddress` provides the same functionality as GetProcedureAddress (1504), and is provided for compatibility with Delphi.

75.12.148 GetProcedureAddress

Synopsis: Get the address of a procedure or symbol in a dynamic library.

Declaration:
```
function GetProcedureAddress(Lib: TLibHandle;
                             const ProcName: AnsiString) : Pointer
function GetProcedureAddress(Lib: TLibHandle; Ordinal: TOrdinalEntry)
                             : Pointer
```

Visibility: default

Description: `GetProcedureAddress` returns a pointer to the location in memory of the symbol `ProcName` or ordinal value `Ordinal` in the dynamically loaded library specified by it's handle `lib`. If the symbol cannot be found or the handle is invalid, `Nil` is returned.

On Windows, only an exported procedure or function can be searched this way. On Unix platforms the location of any exported symbol can be retrieved this way.

Only windows and OS/2 support getting the address of a function using an ordinal value.

Errors: If the symbol cannot be found, `Nil` is returned.

See also: LoadLibrary (1531), UnLoadLibrary (1594)

75.12.149 GetProcessID

Synopsis: Get the current process ID.

Declaration:
```
function GetProcessID : SizeUInt
```

Visibility: default

Description: `GetProcessID` returns the current process ID. The meaning of the return value of this call is system dependent.

Errors: None.

See also: GetThreadID (1505)

75.12.150 GetResourceManager

Synopsis: Return the currently active resource manager.

Declaration:
```
procedure GetResourceManager(var Manager: TResourceManager)
```

Visibility: default

Description: `GetResourceManager` returns the currently active resource manager record in `Manager`. There is always an active resource manager record.

Errors: None.

See also: TResourceManager (1423), SetResourceManager (1569)

75.12.151 GetSSECSR

Declaration:
```
function GetSSECSR : DWord
```

Visibility: default

75.12.152 GetTextCodePage

Synopsis: Get the codepage used in a text file.

Declaration: `function GetTextCodePage(var T: Text) : TSystemCodePage`

Visibility: default

Description: `GetTextCodePage` returns the codepage that the text file `T` uses. All strings written to the file will be converted to the indicated codepage. By default, the codepage is set to `CP_ACP`.

Errors: None.

See also: TextRec (1416), SetTextCodePage (1571)

75.12.153 GetThreadID

Synopsis: Get the current Thread ID.

Declaration: `function GetThreadID : TThreadID`

Visibility: default

Description: `GetThreadID` returns the current process ID. The meaning of the return value of this call is system dependent.

See also: GetProcessID (1504)

75.12.154 GetThreadManager

Synopsis: Return the current thread manager.

Declaration: `function GetThreadManager(var TM: TThreadManager) : Boolean`

Visibility: default

Description: `GetThreadManager` returns the currently used thread manager in `TM`.

For more information about thread programming, see the programmer's guide.

See also: SetThreadManager (1572), TThreadManager (1428)

75.12.155 GetTypeKind

Synopsis: Return type kind for a type.

Declaration: `function GetTypeKind(const T: AnyType) : TTypeKind`

Visibility: default

Description: `GetTypeKind` is a compiler intrinsic: it returns the type kind for the type `T`. In difference with the TypeInfo (1591), if no type information was yet generated for the type, this statement will not ensure that the type information is available: the compiler knows the correct value, and will directly insert it into the code as a constant.

See also: Default (1476), TypeInfo (1591), TypeOf (1591), Initialize (1517), Finalize (1495)

75.12.156 GetUnicodeStringManager

Synopsis: Return a copy of the currently active UnicodeString manager.

Declaration: `procedure GetUnicodeStringManager(var Manager: TUnicodeStringManager)`

Visibility: default

Description: `GetUnicodeStringManager` returns a copy of the currently active Unicode string manager in `Old`

UnicodeStrings are implemented in different ways on different platforms. Therefore, the Free Pascal Runtime library has no fixed implementation of widestring routines. Instead, it defines a UnicodeStringManager record, with callbacks that can be set to an implementation which is most efficient on the current platform. On windows, standard Windows routines will be used. On Unix and Linux, an implementation based on the C library is available (in unit cwstring).

It is possible to implement a custom unicodestring manager, optimized for the current application, without having to recompile the complete Run-Time Library.

See also: SetUnicodeStringManager (1572), TUnicodeStringManager (1431)

75.12.157 GetVariantManager

Synopsis: Return the current variant manager.

Declaration: `procedure GetVariantManager(var VarMgr: tvariantmanager)`

Visibility: default

Description: `GetVariantManager` returns the current variant manager in `varmgr`.

See also: SetVariantManager (1572)

75.12.158 GetWideStringManager

Synopsis: Return a copy of the currently active widestring manager.

Declaration: `procedure GetWideStringManager(var Manager: TUnicodeStringManager)`

Visibility: default

Description: `GetWideStringManager` returns a copy of the currently active heap manager in `Old`

WideStrings are implemented in different ways on different platforms. Therefore, the Free Pascal Runtime library has no fixed implementation of widestring routines. Instead, it defines a WideString manager record, with callbacks that can be set to an implementation which is most efficient on the current platform. On windows, standard Windows routines will be used. On Unix and Linux, an implementation based on the C library is available (in unit cwstring).

It is possible to implement a custom widestring manager, optimized for the current application, without having to recompile the complete Run-Time Library.

See also: SetWideStringManager (1573), TWideStringManager (1440)

75.12.159 get_caller_addr

Synopsis: Return the address of the caller.

Declaration:
```
function get_caller_addr(framebp: pointer; addr: CodePointer)
                                 : CodePointer
```

Visibility: default

Description: `get_caller_frame` returns a pointer to address (the return address) of the caller of the routine which has as frame `framebp`.

See also: get_frame (1508), get_caller_frame (1507), Dump_Stack (1481)

75.12.160 get_caller_frame

Synopsis: Return the frame pointer of the caller.

Declaration:
```
function get_caller_frame(framebp: pointer; addr: CodePointer) : pointer
```

Visibility: default

Description: `get_caller_frame` returns a pointer to the frame of the caller of the routine which has as frame `framebp`.

See also: get_caller_addr (1507), get_frame (1508), Dump_Stack (1481)

75.12.161 get_caller_stackinfo

Synopsis: Return caller stack infomation.

Declaration:
```
procedure get_caller_stackinfo(var framebp: pointer;
                               var addr: CodePointer)
```

Visibility: default

Description: `get_caller_stackinfo` returns caller address in `addr` and frame base pointer in `framebp`.

See also: CaptureBacktrace (1464), Get_pc_addr (1508), get_caller_addr (1507), get_caller_frame (1507)

75.12.162 get_cmdline

Synopsis: Return the command-line as a null-terminated string.

Declaration:
```
function get_cmdline : PChar
```

Visibility: default

Description: `get_cmdline` returns the complete command-line as a null-terminated string. It is not recommended to use this function, since it builds a complete value from the actual command-line arguments. Instead, ParamCount (1541) and ParamStr (1542) should be used.

See also: ParamCount (1541), ParamStr (1542)

75.12.163 get_frame

Synopsis: Return the current frame.

Declaration: `function get_frame : pointer`

Visibility: default

Description: `get_frame` returns a pointer to the current stack frame.

See also: get_caller_addr (1507), get_caller_frame (1507)

75.12.164 Get_pc_addr

Synopsis: Get Program Counter address.

Declaration: `function Get_pc_addr : CodePointer`

Visibility: default

Description: `Get_pc_addr` returns the program counter address (current execution address).

See also: CaptureBacktrace (1464), get_caller_stackinfo (1507), get_caller_addr (1507), get_caller_frame (1507)

75.12.165 greaterthan(variant,variant):Boolean

Synopsis: Implement > (greater than) operation on variants.

Declaration: `operator >(const op1: variant; const op2: variant) : Boolean`

Visibility: default

Description: The implementation of the "greater than" comparison `(>)` operation is delegated to the variant manager with operation `opcmpgt`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator <(variant, variant): boolean (1362)

75.12.166 greaterthanorequal(variant,variant):Boolean

Synopsis: Implement >= (greater than or equal) operation on variants.

Declaration: `operator >=(const op1: variant; const op2: variant) : Boolean`

Visibility: default

Description: The implementation of the "greater than or equal" comparison `(>=)` operation is delegated to the variant manager with operation `opcmpge`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator <(variant, variant): boolean (1362)

75.12.167 Halt

Synopsis: Stop program execution.

Declaration:
```
procedure Halt(errnum: LongInt)
procedure Halt
```

Visibility: default

Description: `Halt` stops program execution and returns control to the calling program. The optional argument `Errnum` specifies an exit value. If omitted, zero is returned.

Note that this skips any try/finally (implicit or explicit) or try/except blocks, thus may result in memory leaks. Finalization sections of units will be executed.

Errors: None.

See also: Exit (1488)

Listing: ./examples/refex/ex30.pp

```
Program Example30;

{ Program to demonstrate the Halt function. }

begin
 WriteLn ('Before Halt.');
 Halt (1); { Stop with exit code 1 }
 WriteLn ('After Halt doesn''t get executed.');
end.
```

75.12.168 HexStr

Synopsis: Convert integer value to string with hexadecimal representation.

Declaration:
```
function HexStr(Val: LongInt; cnt: Byte) : shortstring
function HexStr(Val: Int64; cnt: Byte) : shortstring
function HexStr(Val: QWord; cnt: Byte) : shortstring
function HexStr(Val: Pointer) : shortstring
```

Visibility: default

Description: `HexStr` returns a string with the hexadecimal representation of `Value`. The string has exactly `cnt` characters. (i.e. only the `cnt` rightmost nibbles are taken into account) To have a complete representation of a Longint-type value, 8 nibbles are needed, i.e. `cnt=8`.

Errors: None.

See also: Str (1577), Val (1598), BinStr (1458)

Listing: ./examples/refex/ex81.pp

```
Program example81;

{ Program to demonstrate the HexStr function }

Const Value = 45678;

Var I : longint;
```

```
begin
  For I:=1 to 10 do
    WriteIn (HexStr(Value,I));
end.
```

75.12.169 Hi

Synopsis: Return high byte/word/nibble of value.

Declaration:
```
function Hi(b: Byte) : Byte
function Hi(i: Integer) : Byte
function Hi(w: Word) : Byte
function Hi(l: LongInt) : Word
function Hi(l: DWord) : Word
function Hi(i: Int64) : DWord
function Hi(q: QWord) : DWord
```

Visibility: default

Description: `Hi` returns the high nibble, byte or word or longword from `X`, depending on the size of X.

Table 75.21:

Size	Return value
8	Byte, High nibble
16	Byte, High byte
32	Word, High word
64	Cardinal, High DWord

Note that in Delphi or TP, this function always treats its argument as if it was a Word, so the results may differ from FPC.

Errors: None.

See also: Lo (1530)

Listing: ./examples/refex/ex31.pp

```
Program Example31;

{ Program to demonstrate the Hi function. }

var
  L : Longint;
  W : Word;
  B : Byte;

begin
  L:=1 Shl 16;      { = $10000 }
  W:=1 Shl 8;       { = $100 }
  B:=1 Shl 4;       { = $10  }
  WriteIn (Hi(L)); { Prints 1 }
  WriteIn (Hi(W)); { Prints 1 }
```

```
  WriteLn (Hi(B)); { Prints 1 }
end.
```

75.12.170 High

Synopsis: Return highest index of open array or enumerated.

Declaration: `function High(Arg: TypeOrVariable) : TOrdinal`

Visibility: default

Description: The return value of `High` depends on it's argument:

1. If the argument is an ordinal type, `High` returns the highest value in the range of the given ordinal type.
2. If the argument is an array type or an array type variable then `High` returns the highest possible value of it's index. For dynamic arrays, it returns the same as `Length -1`, meaning that it reports -1 for empty arrays.
3. If the argument is an open array identifier in a function or procedure, then `High` returns the highest index of the array, as if the array has a zero-based index. If the array is empty, then -1 is returned.
4. If the argument is a set type then it returns the highest value of the underlying ordinal type.

The return type is always the same type as the type of the argument (This can lead to some nasty surprises !).

Errors: None.

See also: Low (1534), Ord (1540), Pred (1545), Succ (1582)

Listing: ./examples/refex/ex80.pp

```
Program example80;

{ Example to demonstrate the High and Low functions. }

Type TEnum = ( North, East, South, West );
     TRange = 14..55;
     TArray = Array [2..10] of Longint;

Function Average (Row : Array of Longint) : Real;

Var I : longint;
    Temp : Real;


begin
  Temp := Row[0];
  For I := 1 to High(Row) do
    Temp := Temp + Row[i];
  Average := Temp / (High(Row)+1);
end;

Var A : TEnum;
    B : TRange;
    C : TArray;
```

```
    I : longint;

begin
  Writeln ('TEnum  goes from : ',Ord(Low(TEnum)),' to ', Ord(high(TEnum)),'.');
  Writeln ('A      goes from : ',Ord(Low(A)),' to ', Ord(high(A)),'.');
  Writeln ('TRange goes from : ',Ord(Low(TRange)),' to ', Ord(high(TRange)),'.');
  Writeln ('B      goes from : ',Ord(Low(B)),' to ', Ord(high(B)),'.');
  Writeln ('TArray index goes from : ',Ord(Low(TArray)),' to ', Ord(high(TArray)),'.');
  Writeln ('C index      goes from : ',Low(C),' to ', high(C),'.');
  For I:=Low(C) to High(C) do
    C[i]:=I;
  Writeln ('Average :',Average(c));
  Write ('Type of return value is always same as type of argument:');
  Writeln(high(high(word)));
end.
```

75.12.171 HINSTANCE

Synopsis: Windows compatibility type for use in resources.

Declaration: `function HINSTANCE : TFPResourceHMODULE`

Visibility: default

Description: This is an opaque type.

75.12.172 Inc

Synopsis: Increase value of integer variable.

Declaration:
```
procedure Inc(var X: TOrdinal)
procedure Inc(var X: TOrdinal; Increment: TOrdinal)
```

Visibility: default

Description: `Inc` increases the value of `X` with `Increment`. If `Increment` isn't specified, then 1 is taken as a default.

`Inc` can be used on typed pointers: in that case it increases the value with `Increment` the size of the type the pointer points to. This works independently of the setting of the `$POINTERMATH` directive.

Errors: If range checking is on, then A range check can occur, or an overflow error, when an attempt is made to increase `X` over its maximum value.

See also: Dec (1475)

Listing: ./examples/refex/ex32.pp

```
Program Example32;

{ Program to demonstrate the Inc function. }

Const
  C : Cardinal  = 1;
  L : Longint   = 1;
  I : Integer   = 1;
  W : Word      = 1;
```

```
  B : Byte      = 1;
  SI : ShortInt = 1;
  CH : Char     = 'A';

begin
  Inc (C);      { C:=2    }
  Inc (L,5);    { L:=6    }
  Inc (I,-3);   { I:=-2   }
  Inc (W,3);    { W:=4    }
  Inc (B,100);  { B:=101  }
  Inc (SI,-3);  { Si:=-2  }
  Inc (CH,1);   { ch:='B' }
end.
```

75.12.173 Include

Synopsis: Include element in set if it was not yet present.

Declaration: `procedure Include(var S: TSetType; E: TSetElement)`

Visibility: default

Description: `Include` includes `E` in the set `S` if it is not yet part of the set. E should be of the same type as the base type of the set `S`.

Thus, the two following statements do the same thing:

```
S:=S+[E];
Include(S,E);
```

For an example, see Exclude (1487)

Errors: If the type of the element `E` is not equal to the base type of the set `S`, the compiler will generate an error.

See also: Exclude (1487)

75.12.174 IndexByte

Synopsis: Search for a byte in a memory range.

Declaration: `function IndexByte(const buf; len: SizeInt; b: Byte) : SizeInt`

Visibility: default

Description: `IndexByte` searches the memory at `buf` for maximally `len` positions for the byte `b` and returns it's position if it found one. If `b` is not found then -1 is returned. The position is zero-based.

Errors: `Buf` and `Len` are not checked to see if they are valid values.

See also: IndexChar (1514), IndexDWord (1515), IndexWord (1516), CompareByte (1466)

Listing: ./examples/refex/ex105.pp

```
Program Example105;

{ Program to demonstrate the IndexByte function. }

Const
  ArraySize = 256;
  MaxValue = 256;

Var
  Buffer : Array[1..ArraySize] of Byte;
  I,J : longint;
  K : Byte;

begin
  Randomize;
  For I:=1 To ArraySize do
    Buffer[I]:=Random(MaxValue);
  For I:=1 to 10 do
    begin
    K:=Random(MaxValue);
    J:=IndexByte(Buffer,ArraySize,K);
    if J=-1 then
      Writeln('Value ',K,' was not found in buffer.')
    else
      Writeln('Found ',K,' at position ',J,' in buffer');
    end;
end.
```

75.12.175 IndexChar

Synopsis: Search for a character in a memory range.

Declaration: `function IndexChar(const buf; len: SizeInt; b: Char) : SizeInt`

Visibility: default

Description: `IndexChar` searches the memory at `buf` for maximally `len` positions for the character `b` and returns it's position if it found one. If `b` is not found then -1 is returned. The position is zero-based. The `IndexChar0` variant stops looking if a null character is found, and returns -1 in that case.

Errors: `Buf` and `Len` are not checked to see if they are valid values.

See also: IndexByte (1513), IndexDWord (1515), IndexWord (1516), CompareChar (1467)

Listing: ./examples/refex/ex108.pp

```
Program Example108;

{ Program to demonstrate the IndexChar function. }

Const
  ArraySize = 1000;
  MaxValue = 26;

Var
  Buffer : Array[1..ArraySize] of Char;
  I,J : longint;
```

```
  K : Char;

begin
  Randomize;
  For I:=1 To ArraySize do
    Buffer[I]:=chr(Ord('A')+Random(MaxValue));
  For I:=1 to 10 do
    begin
    K:=chr(Ord('A')+Random(MaxValue));
    J:=IndexChar(Buffer,ArraySize,K);
    if J=-1 then
      Writeln('Value ',K,' was not found in buffer.')
    else
      Writeln('Found ',K,' at position ',J,' in buffer');
    end;
end.
```

75.12.176 IndexChar0

Synopsis: Return index of a character in null-terminated array of char.

Declaration: `function IndexChar0(const buf; len: SizeInt; b: Char) : SizeInt`

Visibility: default

Description: `IndexChar0` returns the index of the character `b` in the null-terminated array `Buf`. At most `len` characters will be searched, or the null character if it is encountered first. If the character is not found, -1 is returned.

Errors: On error, -1 is returned.

See also: IndexByte (1513), IndexChar (1514), IndexWord (1516), IndexDWord (1515), CompareChar0 (1469)

75.12.177 IndexDWord

Synopsis: Search for a DWord value in a memory range.

Declaration: `function IndexDWord(const buf; len: SizeInt; b: DWord) : SizeInt`

Visibility: default

Description: `IndexDWord` searches the memory at `buf` for maximally `len` positions for the DWord `DW` and returns it's position if it found one. If `DW` is not found then -1 is returned. The position is zero-based.

Errors: `Buf` and `Len` are not checked to see if they are valid values.

See also: IndexByte (1513), IndexChar (1514), IndexWord (1516), CompareDWord (1469)

Listing: ./examples/refex/ex106.pp

```
Program Example106;

{ Program to demonstrate the IndexDWord function. }

Const
  ArraySize = 1000;
  MaxValue = 1000;
```

```
Var
  Buffer : Array[1..ArraySize] of DWord;
  I,J : longint;
  K : DWord;

begin
  Randomize;
  For I:=1 To ArraySize do
    Buffer[I]:=Random(MaxValue);
  For I:=1 to 10 do
    begin
    K:=Random(MaxValue);
    J:=IndexDWord(Buffer,ArraySize,K);
    if J=-1 then
      Writeln('Value ',K,' was not found in buffer.')
    else
      Writeln('Found ',K,' at position ',J,' in buffer');
    end;
end.
```

75.12.178 IndexQWord

Synopsis: Return the position of a QWord in a memory range.

Declaration: `function IndexQWord(const buf; len: SizeInt; b: QWord) : SizeInt`

Visibility: default

Description: `IndexQWord` checks the first `len` qwords starting at `Buf`, and returns the position (zero-based) of `b`. If `b` does not appear in the first `len` qwords, then -1 is returned.

Note that the search is done on QWord boundaries, but that the address of `buf` need not be on a QWord boundary.

Errors: No check is done to see whether the indicated memory range is valid. If it is not, a run-error or exception may be triggered.

See also: IndexDWord (1515)

75.12.179 Indexword

Synopsis: Search for a WORD value in a memory range.

Declaration: `function Indexword(const buf; len: SizeInt; b: Word) : SizeInt`

Visibility: default

Description: `IndexWord` searches the memory at `buf` for maximally `len` positions for the Word `W` and returns it's position if it found one. If `W` is not found then -1 is returned.

Errors: `Buf` and `Len` are not checked to see if they are valid values.

See also: IndexByte (1513), IndexDWord (1515), IndexChar (1514), CompareWord (1470)

Listing: ./examples/refex/ex107.pp

```
Program Example107;

{ Program to demonstrate the IndexWord function. }

Const
  ArraySize = 1000;
  MaxValue = 1000;

Var
  Buffer : Array[1..ArraySize] of Word;
  I,J : longint;
  K : Word;

begin
  Randomize;
  For I:=1 To ArraySize do
    Buffer[I]:=Random(MaxValue);
  For I:=1 to 10 do
    begin
    K:=Random(MaxValue);
    J:=IndexWord(Buffer,ArraySize,K);
    if J=-1 then
      Writeln('Value ',K,' was not found in buffer.')
    else
      Writeln('Found ',K,' at position ',J,' in buffer');
    end;
end.
```

75.12.180 InitCriticalSection

Synopsis: Initialize a critical section.

Declaration: `procedure InitCriticalSection(var cs: TRTLCRITICALSECTION)`

Visibility: default

Description: `InitCriticalSection` initializes a critical section `CS` for use. Before using a critical section with EnterCriticalsection (1483) or LeaveCriticalsection (1527) the critical section should be initialized with `InitCriticalSection`.

When a critical section is no longer used, it should be disposed of with DoneCriticalsection (1479)

See also: DoneCriticalsection (1479), EnterCriticalsection (1483), LeaveCriticalsection (1527)

75.12.181 Initialize

Synopsis: Initialize memory block using RTTI.

Declaration: `procedure Initialize(var T: TAnyType; ACount: SizeInt)`

Visibility: default

Description: `Initialize` is a compiler intrinsic: it initializes a memory area belonging to a variable `T` of a managed type (1362) (`TManagedType`). Initializing means zeroing out the memory area. In this sense it is close in functionality to Default (1476), but `Default` returns an already initialized variable, whereas `Initialize` initializes managed types in a previously declared memory area.

`Default` is a function, and one can think of `Initialize` as the procedural version of `Default`.

`Initialize` performs the opposite operation of Finalize (1495), which should be used to clean up the memory block when it is no longer needed.

The optional `ACount` parameter can be used to initialize an array. It then specifies the number of elements in the array.

Note that the function can be used on an already used variable, and that for structured types, only managed fields are initialized.

See also: Finalize (1495), Default (1476), ManagedTypes (1362), TypeInfo (1591)

Listing: ./examples/refex/ex117.pp

```
{
  This example demonstrates the use of the Initialize and Finalize functions
  used to initialize (and clean up) any RTTI-enabled data not allocated with
  New or Create.
}

{$mode objfpc}
{$h+} // use ansistrings, they need to be initialized.
Type
  PData = ^TData;
  TData = record
    Street,City,Zip,Country, Tel: String;
    StreetNumber : Integer;
  end;

var
  Data: PData;

begin
  // Do not use New.
  GetMem(Data,SizeOf(TData));
  Try
    { Initialize the structure in memory, using Run-Time Type Information}
    Initialize(Data^);
    { Assign some string data to the ansistring contents.
      Note that this only works because the record was zeroed out by Initialize}
    Data^.Street:='Sesame Street';
    Data^.City:='Heaven';
    Data^.Zip:='7777777';
    Data^.Country:='Spain';
    Data^.StreetNumber:=3;
  Finally
    { Clean up the record contents.
      Again, the structure of the record is detected through
      Run-time Type Information }
    Finalize(Data^);
    FreeMem(Data);
  end;
end.
```

Listing: ./examples/refex/ex118.pp

```
{
  This example demonstrates the use of the Initialize and Finalize functions
  used to initialize (and clean up) any RTTI-enabled data not allocated with
```

```
  New or Create.
}

{$mode objfpc}
{$h+} // use ansistrings, they need to be initialized.
Type
  PData = ^TData;
  TData = record
    Street,City,Zip,Country, Tel: String;
    StreetNumber : Integer;
  end;

var
  Data: PData;

begin
  // We use the fact that a pointer is also usable as an array.
  GetMem(Data,SizeOf(TData)*2);
  Try
    { Initialize the structure in memory, using Run-Time Type Information}
    Initialize(Data^,2);
    { Assign some string data to the ansistring contents.
      Note that this only works because the record was zeroed out by Initialize}
    Data[0].Street:='Sesame Street';
    Data[0].City:='Heaven';
    Data[0].Zip:='7777777';
    Data[0].Country:='Spain';
    Data[0].StreetNumber:=3;
    // Second, well known street
    Data[1].Street:='Wall Street';
    Data[1].City:='New York';
    Data[1].Zip:='10005';
    Data[1].Country:='USA';
    Data[1].StreetNumber:=11;
  Finally
    { Clean up the record contents.
      Again, the structure of the record is detected through
      Run-time Type Information }
    Finalize(Data^);
    FreeMem(Data);
  end;
end.
```

Listing: ./examples/refex/ex123.pp

```
program ex123;

{
  Example to show that Initialize can be used on a variable that was already used,
  and that only fields of managed type are affected.
}

type
  TTest = object
    I: Integer;
    S: AnsiString;
  end;

procedure Test;
```

```
var
  V: TTest;

begin
  V.I := 10;
  V.S := 'x';
  WriteLn(V.I);
  WriteLn(V.S);
  Initialize(V);
  WriteLn(V.I); // unchanged
  WriteLn(V.S); // empty
end;

begin
  Test;
end.
```

75.12.182 InitializeArray

Synopsis: Initialize managed-type elements in array.

Declaration: `procedure InitializeArray(p: Pointer; typeInfo: Pointer; count: SizeInt)`

Visibility: default

Description: `InitializeArray` initializes managed types in the array pointed to by `p`. For this, it uses the type information of the elements as specified in `typeinfo`.

Under normal circumstances, this procedure should not be used, it is called automatically by the compiler when an array-typed variable is declared and the array contains elements with managed types.

See also: FinalizeArray (1495), CopyArray (1474)

75.12.183 InitThread

Synopsis: Initialize a thread.

Declaration: `procedure InitThread(stklen: SizeUInt)`

Visibility: default

Description: Do not use, this is used internally by the thread manager.

75.12.184 InitThreadVars

Synopsis: Initialize threadvars.

Declaration: `procedure InitThreadVars(RelocProc: TRelocateThreadVarHandler)`

Visibility: default

Description: This routine should be called when threading is started. It is called by the compiler and should never be called manually, only from a thread manager.

Errors: None.

See also: TThreadManager (1428)

75.12.185 Insert

Synopsis: Insert one string or dynamic array in another.

Declaration:
```
procedure Insert(const source: string; var S: string;
                 const Index: Integer)
procedure Insert(const source: DynaArrayType; var S: DynArrayType;
                 const Index: Integer)
```

Visibility: default

Description: `Insert` inserts string `Source` in string `S`, at position `Index`, shifting all characters after `Index` to the right. The resulting string is truncated at 255 characters, if needed. (i.e. for shortstrings)

`Index` is a 1-based index. if `Index` is less than 1, the insert of `Source` happens at the start of the string, as if the value 1 was specified.

If the value of `Index` is larger than the length of the string `S`, `Source` is appended to the string `S`.

For dynamic arrays, `Insert` inserts the elements of array `Source` in array `S`, at position `Index`, shifting all elements after `Index` to the right.

`Index` is a 0-based index. if `Index` is less than 0, the insert of `Source` happens at position0.

If the value of `Index` is larger than the length of the array `S`, `Source` is appended to the array `S`.

Errors: None.

See also: Delete (1477), Copy (1473), Pos (1543)

Listing: ./examples/refex/ex33.pp

```
Program Example33;

{ Program to demonstrate the Insert function. }

Var S : String;

begin
  S:='Free Pascal is difficult to use !';
  Insert ('NOT ',S,pos('difficult',S));
  writeln (s);
end.
```

75.12.186 Int

Synopsis: Calculate integer part of floating point value.

Declaration:
```
function Int(d: ValReal) : ValReal
```

Visibility: default

Description: `Int` returns the integer part of any Real `X`, as a Real.

Errors: None.

See also: Frac (1499), Round (1557)

Listing: ./examples/refex/ex34.pp

```
Program Example34;

{ Program to demonstrate the Int function. }

begin
  WriteLn (Int(123.456):0:1);  { Prints  123.0 }
  WriteLn (Int(-123.456):0:1); { Prints -123.0 }
end.
```

75.12.187 intdivide(variant,variant):variant

Synopsis: Implement `div` (integer division) operation on variants.

Declaration:
```
operator div(const op1: variant; const op2: variant) : variant
```

Visibility: default

Description: The implementation of the integer division `Div` operation is delegated to the variant manager with operation `opintdivide`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator mod(variant, variant): variant (1362)

75.12.188 InterlockedCompareExchange

Synopsis: Conditional exchange.

Declaration:
```
function InterlockedCompareExchange(var Target: LongInt;
                                    NewValue: LongInt;
                                    Comperand: LongInt) : LongInt
function InterlockedCompareExchange(var Target: Pointer;
                                    NewValue: Pointer;
                                    Comperand: Pointer) : Pointer
function InterlockedCompareExchange(var Target: Cardinal;
                                    NewValue: Cardinal;
                                    Comperand: Cardinal) : Cardinal
```

Visibility: default

Description: `InterlockedCompareExchange` does an compare-and-exchange operation on the specified values in a thread-safe way. The function compares `Target` and `Comperand` and exchanges `Target` with `NewValue` if `Target` and `Comperand` are equal. It returns the old value of `Target`. This is done in a thread-safe way, i.e., only one processor is accessing the `Target` variable at a time.

Errors: None.

See also: InterLockedDecrement (1523), InterLockedIncrement (1525), InterLockedExchange (1523), InterLockedExchangeAdd (1524)

75.12.189 InterlockedCompareExchange64

Declaration:
```
function InterlockedCompareExchange64(var Target: Int64;
                                       NewValue: Int64; Comperand: Int64)
                                        : Int64
function InterlockedCompareExchange64(var Target: QWord;
                                       NewValue: QWord; Comperand: QWord)
                                        : QWord
```

Visibility: default

75.12.190 InterlockedCompareExchangePointer

Synopsis: Compare pointers in an atomic operation.

Declaration:
```
function InterlockedCompareExchangePointer(var Target: Pointer;
                                            NewValue: Pointer;
                                            Comperand: Pointer) : Pointer
```

Visibility: default

Description: `InterlockedCompareExchangePointer` compares `Comparand` with `Target`. if they are equal, replaces `Target` with `NewValue`. This is done in a single atomic operation.

See also: InterlockedCompareExchange (1522)

75.12.191 InterlockedDecrement

Synopsis: Thread-safe decrement.

Declaration:
```
function InterlockedDecrement(var Target: LongInt) : LongInt
function InterlockedDecrement(var Target: Pointer) : Pointer
function InterlockedDecrement(var Target: Cardinal) : Cardinal
```

Visibility: default

Description: `InterLockedDecrement` decrements `Target` with 1 and returns the result. This is done in a thread-safe way. (i.e. only one processor is accessing the variable at a time).

Errors: None.

See also: InterLockedIncrement (1525), InterLockedExchange (1523), InterLockedExchangeAdd (1524), InterlockedCompareExchange (1522)

75.12.192 InterlockedDecrement64

Declaration:
```
function InterlockedDecrement64(var Target: Int64) : Int64
function InterlockedDecrement64(var Target: QWord) : QWord
```

Visibility: default

75.12.193 InterlockedExchange

Synopsis: Exchange 2 integers in a thread-safe way.

Declaration:
```
function InterlockedExchange(var Target: LongInt; Source: LongInt)
                                  : LongInt
function InterlockedExchange(var Target: Pointer; Source: Pointer)
                                  : Pointer
function InterlockedExchange(var Target: Cardinal; Source: Cardinal)
                                  : Cardinal
```

Visibility: default

Description: `InterLockedExchange` stores `Source` in `Target` and returns the old value of `Target`. This is done in a thread-safe way, i.e., only one processor is accessing the `Target` variable at a time.

Errors: None.

See also: InterLockedDecrement (1523), InterLockedIncrement (1525), InterLockedExchangeAdd (1524), InterlockedCompareExchange (1522)

75.12.194 InterlockedExchange64

Declaration:
```
function InterlockedExchange64(var Target: Int64; Source: Int64) : Int64
function InterlockedExchange64(var Target: QWord; Source: QWord) : QWord
```

Visibility: default

75.12.195 InterlockedExchangeAdd

Synopsis: Thread-safe add and exchange of 2 values.

Declaration:
```
function InterlockedExchangeAdd(var Target: LongInt; Source: LongInt)
                                    : LongInt
function InterlockedExchangeAdd(var Target: Pointer; Source: Pointer)
                                    : Pointer
function InterlockedExchangeAdd(var Target: Cardinal; Source: Cardinal)
                                    : Cardinal
```

Visibility: default

Description: `InterLockedExchangeAdd` adds to `Target` the value of `Source` in a thread-safe way, and returns the old value of `Target`.This is done in a thread-safe way, i.e., only one processor is accessing the `Target` variable at a time.

Errors: None.

See also: InterLockedDecrement (1523), InterLockedIncrement (1525), InterLockedExchange (1523), InterlockedCompareExchange (1522)

75.12.196 InterlockedExchangeAdd64

Declaration:
```
function InterlockedExchangeAdd64(var Target: Int64; Source: Int64)
                                      : Int64
function InterlockedExchangeAdd64(var Target: QWord; Source: QWord)
                                      : QWord
```

Visibility: default

75.12.197 InterlockedIncrement

Synopsis: Thread-safe increment.

Declaration:
```
function InterlockedIncrement(var Target: LongInt) : LongInt
function InterlockedIncrement(var Target: Pointer) : Pointer
function InterlockedIncrement(var Target: Cardinal) : Cardinal
```

Visibility: default

Description: `InterLockedIncrement` increments `Target` with 1 and returns the result. This is done in a thread-safe way (i.e. only one processor is accessing the variable at a time).

Errors: None.

See also: InterLockedDecrement (1523), InterLockedExchange (1523), InterLockedExchangeAdd (1524), InterlockedCompareExchange (1522)

75.12.198 InterlockedIncrement64

Declaration:
```
function InterlockedIncrement64(var Target: Int64) : Int64
function InterlockedIncrement64(var Target: QWord) : QWord
```

Visibility: default

75.12.199 IOResult

Synopsis: Return result of last file IO operation.

Declaration:
```
function IOResult : Word
```

Visibility: default

Description: IOresult contains the result of any input/output call, when the `{\$i-}` compiler directive is active, disabling IO checking. When the flag is read, it is reset to zero. If `IOresult` is zero, the operation completed successfully. If non-zero, an error occurred. The following errors can occur:

dos errors :

2File not found.

3Path not found.

4Too many open files.

5Access denied.

6Invalid file handle.

12Invalid file-access mode.

15Invalid disk number.

16Cannot remove current directory.

17Cannot rename across volumes.

I/O errors :

100 Error when reading from disk.

101 Error when writing to disk.

102 File not assigned.

103 File not open.
104 File not opened for input.
105 File not opened for output.
106 Invalid number.

Fatal errors :

150 Disk is write protected.
151 Unknown device.
152 Drive not ready.
153 Unknown command.
154CRC check failed.
155 Invalid drive specified..
156 Seek error on disk.
157 Invalid media type.
158 Sector not found.
159 Printer out of paper.
160 Error when writing to device.
161 Error when reading from device.
162 Hardware failure.

Errors: None.

Listing: ./examples/refex/ex35.pp

```
Program Example35;

{ Program to demonstrate the IOResult function. }

Var F : text;

begin
  Assign (f,paramstr(1));
  {$i-}
  Reset (f);
  {$i+}
  If IOresult<>0 then
    writeln ('File ',paramstr(1),' doesn''t exist')
  else
    writeln ('File ',paramstr(1),' exists');
end.
```

75.12.200 IsDynArrayRectangular

Synopsis: Check whether all dimensions have the same size.

Declaration: `function IsDynArrayRectangular(a: Pointer; typeInfo: Pointer) : Boolean`

Visibility: default

Description: `IsDynArrayRectangular` returns `True` if all dimensions of the dynamic array `a` with type information `typinfo` have the same bounds. It returns `True` if the array is empty.

See also: InitializeArray (1520), FinalizeArray (1495), CopyArray (1474), DynArraySize (1482), DynArrayClear (1481), DynArrayBounds (1481), DynArrayDim (1481)

75.12.201 IsMemoryManagerSet

Synopsis: Is the memory manager set.

Declaration: `function IsMemoryManagerSet : Boolean`

Visibility: default

Description: `IsMemoryManagerSet` will return `True` if the memory manager has been set to another value than the system heap manager, it will return `False` otherwise.

Errors: None.

See also: SetMemoryManager (1567), GetMemoryManager (1503)

75.12.202 Is_IntResource

Synopsis: Check whether a resource is an internal resource.

Declaration: `function Is_IntResource(aStr: PChar) : Boolean`

Visibility: default

Description: `Is_IntResource` returns `True` if the resource type is internal (system predefined) resource or false if it is a user-defined resource type.

Errors: None.

75.12.203 KillThread

Synopsis: Kill a running thread.

Declaration: `function KillThread(threadHandle: TThreadID) : DWord`

Visibility: default

Description: `KillThread` causes a running thread to be aborted. The thread is identified by it's handle or ID `threadHandle`.

The function returns zero if successful. A nonzero return value indicates failure.

Errors: If a failure occurred, a nonzero result is returned. The meaning is system dependent.

See also: WaitForThreadTerminate (1600), EndThread (1483), SuspendThread (1583)

75.12.204 LazyInitThreading

Declaration: `procedure LazyInitThreading`

Visibility: default

75.12.205 LeaveCriticalSection

Synopsis: Leave a critical section.

Declaration: `procedure LeaveCriticalSection(var cs: TRTLCRITICALSECTION)`

Visibility: default

Description: `LeaveCriticalSection` signals that the current thread is exiting the critical section `CS` it has entered with EnterCriticalSection (1483).

The critical section must have been initialized with InitCriticalSection (1517) prior to a call to `EnterCriticalsection` and `LeaveCriticalSection`.

See also: InitCriticalsection (1517), DoneCriticalsection (1479), EnterCriticalsection (1483)

75.12.206 leftshift(variant,variant):variant

Synopsis: Implement binary `shl` operation on variants.

Declaration: `operator shl(const op1: variant; const op2: variant) : variant`

Visibility: default

Description: The implementation of the `shl` operation is delegated to the variant manager with operation `opshiftleft`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator shr(variant, variant): variant (1362)

75.12.207 Length

Synopsis: Returns length of a string or array.

Declaration:
```
function &Length(S: AStringType) : SizeInt
function &Length(A: DynArrayType) : SizeInt
```

Visibility: default

Description: `Length` returns the length of the string or array `S`, which is limited to 255 for shortstrings. If the string `S` is empty, 0 is returned.

Note: The length of the string `S` is stored in `S[0]` for shortstrings only. The `Length` function should always be used on ansistrings and widestrings.

For dynamic or static arrays, the function returns the number of elements in the array.

`Length` also supports arguments of type `PChar`and `PWideChar`, in which case it is identical to the `StrLen` and `WStrLen` functions, respectively. In this case, the function actually calculates the length of the null-terminated string, and its execution time is proportional to the string length because the terminating null character is searched through a linear scan.

Errors: None.

See also: Pos (1543), SetLength (1566)

Listing: ./examples/refex/ex36.pp

```
Program Example36;

{ Program to demonstrate the Length function. }

type
  somebytes = array [6..10] of byte;
  somewords = array [3..10] of word;
```

```
Var
  S : String;
  I : Integer;
  bytes : somebytes;
  words : somewords;

begin
  S:='';
  for i:=1 to 10 do
    begin
    S:=S+'*';
    Writeln (Length(S):2,' : ',s);
    end;
  Writeln('Bytes : ',length(bytes));
  Writeln('Words : ',length(words));
end.
```

75.12.208 lessthan(variant,variant):Boolean

Synopsis: Implement < (less than) operation on variants.

Declaration: `operator <(const op1: variant; const op2: variant) : Boolean`

Visibility: default

Description: The implementation of the "less than" comparison `(<)` operation is delegated to the variant manager with operation `opcmplt`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator >(variant, variant): boolean (1362)

75.12.209 lessthanorequal(variant,variant):Boolean

Synopsis: Implement <= (less than or equal) operation on variants.

Declaration: `operator <=(const op1: variant; const op2: variant) : Boolean`

Visibility: default

Description: The implementation of the "less than or equal" comparison `(<=)` operation is delegated to the variant manager with operation `opcmple`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator <(variant, variant): boolean (1362)

75.12.210 LEtoN

Synopsis: Convert Little Endian-ordered integer to Native-ordered integer.

Declaration:
```
function LEtoN(const AValue: SmallInt) : SmallInt
function LEtoN(const AValue: Word) : Word
function LEtoN(const AValue: LongInt) : LongInt
```

```
function LEtoN(const AValue: DWord) : DWord
function LEtoN(const AValue: Int64) : Int64
function LEtoN(const AValue: QWord) : QWord
```

Visibility: default

Description: LEToN will rearrange the bytes in a Little-Endian number to the native order for the current processor. That is, for a little-endian processor, it will do nothing, and for a big-endian processor, it will invert the order of the bytes.

See also: BEtoN (1458), NtoBE (1538), NtoLE (1538)

75.12.211 Ln

Synopsis: Calculate logarithm.

Declaration: `function Ln(d: ValReal) : ValReal`

Visibility: default

Description: Ln returns the natural logarithm of the Real parameter X. X must be positive.

Errors: An run-time error will occur when X is negative.

See also: Exp (1490)

Listing: ./examples/refex/ex37.pp

```
Program Example37;

{ Program to demonstrate the Ln function. }

begin
  WriteLn (Ln(1));       { Prints 0 }
  WriteLn (Ln(Exp(1))); { Prints 1 }
end.
```

75.12.212 Lo

Synopsis: Return low nibble/byte/word of value.

Declaration:
```
function Lo(B: Byte) : Byte
function Lo(i: Integer) : Byte
function Lo(w: Word) : Byte
function Lo(l: LongInt) : Word
function Lo(l: DWord) : Word
function Lo(i: Int64) : DWord
function Lo(q: QWord) : DWord
```

Visibility: default

Description: Lo returns the high nibble, byte or word or longword from X, depending on the size of X.

Table 75.22:

Size	Return value
8	Byte, low nibble
16	Byte, low byte
32	Word, low word
64	Cardinal, low DWord

Errors: None.

See also: Ord (1540), Chr (1465), Hi (1510)

Listing: ./examples/refex/ex38.pp

```
Program Example38;

{ Program to demonstrate the Lo function. }

Var L : Longint;
    W : Word;
    B : Byte;
begin
  L:=(1 Shl 16) + (1 Shl 4);   { $10010 }
  Writeln (Lo(L));             { Prints 16 }
  W:=(1 Shl 8) + (1 Shl 4);    { $110   }
  Writeln (Lo(W));             { Prints 16 }
  B:=$EF;
  Writeln (Lo(B));             { Prints 15 }
end.
```

75.12.213 LoadLibrary

Synopsis: Load a dynamic library and return a handle to it.

Declaration:
```
function LoadLibrary(const Name: RawByteString) : TLibHandle
function LoadLibrary(const Name: UnicodeString) : TLibHandle
```

Visibility: default

Description: `LoadLibrary` loads a dynamic library in file `Name` and returns a handle to it. If the library cannot be loaded, NilHandle (1386) is returned.

No assumptions should be made about the location of the loaded library if a relative pathname is specified. The behaviour is dependent on the platform. Therefore it is best to specify an absolute pathname if possible.

Errors: On error, NilHandle (1386) is returned.

See also: UnloadLibrary (1594), GetProcedureAddress (1504)

75.12.214 LoadResource

Synopsis: Load a resource for use.

Declaration: `function LoadResource(ModuleHandle: TFPResourceHMODULE; ResHandle: TFPResourceHandle) : TFPResourceHGLOBAL`

Visibility: default

Description: `LoadResource` loads a resource identified by `ResHandle` from a module identified by `ModuleHandle`into memory. It returns a handle to the resource.

Loaded resources must be unloaded again using the FreeResource (1500) function.

Errors: On error, 0 is returned.

See also: FindResource (1496), FreeResource (1500), SizeofResource (1574), LockResource (1532), UnlockResource (1594), FreeResource (1500)

75.12.215 LockResource

Synopsis: Lock a resource.

Declaration: `function LockResource(ResData: TFPResourceHGLOBAL) : Pointer`

Visibility: default

Description: `LockResource` locks a resource previously loaded by `LoadResource` into memory. This means that any attempt to modify the resource will fail while it is locked. The function returns a pointer to the resource location in memory.

The resource can be freed again using the UnlockResource (1594) function.

Errors: if the function fails, `Nil` is returned.

See also: FindResource (1496), FreeResource (1500), SizeofResource (1574), LoadResource (1531), UnlockResource (1594), FreeResource (1500)

75.12.216 logicaland(variant,variant):variant

Synopsis: Implement logical/binary `and` operation on variants.

Declaration: `operator and(const op1: variant; const op2: variant) : variant`

Visibility: default

Description: The implementation of the `and` operation is delegated to the variant manager with operation `opand`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator or(variant, variant): variant (1362), operator xor(variant, variant): variant (1362), operator not(variant): variant (1362)

75.12.217 logicalnot(variant):variant

Synopsis: Implement logical/binary `not` operation on variants.

Declaration: `operator not(const op: variant) : variant`

Visibility: default

Description: The implementation of the `not` operation is delegated to the variant manager with operation `opnot`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator and(variant, variant): variant (1362), operator or(variant, variant): variant (1362), operator xor(variant, variant): variant (1362)

75.12.218 logicalor(variant,variant):variant

Synopsis: Implement logical/binary `or` operation on variants.

Declaration: `operator or(const op1: variant; const op2: variant) : variant`

Visibility: default

Description: The implementation of the `or` operation is delegated to the variant manager with operation `opor`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator and(variant, variant): variant (1362), operator xor(variant, variant): variant (1362), operator not(variant): variant (1362)

75.12.219 logicalxor(variant,variant):variant

Synopsis: Implement logical/binary `xor` operation on variants.

Declaration: `operator xor(const op1: variant; const op2: variant) : variant`

Visibility: default

Description: The implementation of the `xor` operation is delegated to the variant manager with operation `opxor`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator or(variant, variant): variant (1362), operator and(variant, variant): variant (1362), operator not(variant): variant (1362)

75.12.220 longjmp

Synopsis: Jump to address.

Declaration: `procedure longjmp(var S: jmp_buf; value: LongInt)`

Visibility: default

Description: `LongJmp` jumps to the address in `S` (a `jmp_buf`), and restores the registers that were stored in it at the corresponding SetJmp (1565) call. In effect, program flow will continue at the `SetJmp` call, which will return `value` instead of 0. If a `value` equal to zero is passed, it will be converted to 1 before passing it on. The call will not return, so it must be used with extreme care. This can be used for error recovery, for instance when a segmentation fault occurred.

For an example, see SetJmp (1565)

Errors: None.

See also: SetJmp (1565)

75.12.221 Low

Synopsis: Return lowest index of open array or enumerated.

Declaration:
```
function Low(Arg: TypeOrVariable) : TOrdinal
```

Visibility: default

Description: The return value of `Low` depends on it's argument:

1. If the argument is an ordinal type, `Low` returns the lowest value in the range of the given ordinal type.
2. If the argument is an array type or an array type variable then `Low` returns the lowest possible value of it's index.
3. If the argument is an open array identifier in a function or procedure, then `Low` returns the lowest element of the array, which is always zero.
4. If the argument is a set type then it returns the lowest value of the underlying ordinal type.

The return type is always the same type as the type of the argument.

for an example, see High (1511).

Errors: None.

See also: High (1511), Ord (1540), Pred (1545), Succ (1582)

75.12.222 LowerCase

Synopsis: Return lowercase version of a string.

Declaration:
```
function LowerCase(const s: shortstring) : shortstring;  Overload
function LowerCase(c: Char) : Char;  Overload
function LowerCase(const s: ansistring) : ansistring
function LowerCase(const s: UnicodeString) : UnicodeString
function LowerCase(c: UnicodeChar) : UnicodeChar
```

Visibility: default

Description: `Lowercase` returns the lowercase version of its argument `C`. If its argument is a string, then the complete string is converted to lowercase. The type of the returned value is the same as the type of the argument.

`Lowercase` does not change the number of characters (or bytes) in an ansistring.

Errors: None.

See also: Upcase (1595)

Listing: ./examples/refex/ex73.pp

```
program Example73;

{ Program to demonstrate the Lowercase function. }

var c:char;

begin
  for c:='A' to 'Z' do
    write(lowercase(c));
```

```
  WriteLn;
  WriteLn(Lowercase('ABCDEFGHIJKLMNOPQRSTUVWXYZ'));
end.
```

75.12.223 MakeLangID

Synopsis: Create a language ID.

Declaration: `function MakeLangID(primary: Word; sub: Word) : Word`

Visibility: default

Description: `MakeLangID` creates a language ID from the `primary` and `sub` language IDS.

75.12.224 MemSize

Synopsis: Return the size of a memory block.

Declaration: `function MemSize(p: pointer) : PtrUInt`

Visibility: default

Description: `MemSize` returns the size of a memory block on the heap.

Errors: Passing an invalid pointer may lead to run-time errors (access violations).

See also: GetMem (1502), FreeMem (1499)

75.12.225 MkDir

Synopsis: Create a new directory.

Declaration:
```
procedure MkDir(const s: shortstring);  Overload
procedure MkDir(const s: RawByteString);  Overload
procedure MkDir(const s: unicodestring);  Overload
```

Visibility: default

Description: `Mkdir` creates a new directory `S`.

For an example, see Rmdir (1554).

Errors: Depending on the state of the `{$I}` switch, a runtime error can be generated if there is an error. In the `{$I-}` state, use `IOResult` to check for errors.

See also: Chdir (1464), Rmdir (1554)

75.12.226 modulus(variant,variant):variant

Synopsis: Implement `mod` (modulo) operation on variants.

Declaration: `operator mod(const op1: variant; const op2: variant) : variant`

Visibility: default

Description: The implementation of the modulo `Mod` operation is delegated to the variant manager with operation `opModulus`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator div(variant, variant): variant (1362)

75.12.227 Move

Synopsis: Move data from one location in memory to another.

Declaration: `procedure Move(const source; var dest; count: SizeInt)`

Visibility: default

Description: `Move` moves `Count` bytes from `Source` to `Dest`.

Errors: If either `Dest` or `Source` is outside the accessible memory for the process, then a run-time error will be generated.

See also: Fillword (1494), Fillchar (1493)

Listing: ./examples/refex/ex42.pp

```
Program Example42;

{ Program to demonstrate the Move function. }

Var S1,S2 : String [30];

begin
  S1:='Hello World !';
  S2:='Bye, bye     !';
  Move (S1,S2,Sizeof(S1));
  Writeln (S2);
end.
```

75.12.228 MoveChar0

Synopsis: Move data till first zero character.

Declaration: `procedure MoveChar0(const buf1; var buf2; len: SizeInt)`

Visibility: default

Description: `MoveChar0` moves `Count` bytes from `buf1` to `buf2`, and stops moving if a zero character is found.

Errors: No checking is done to see if `Count` stays within the memory allocated to the process.

See also: Move (1536)

Listing: ./examples/refex/ex109.pp

```
Program Example109;

{ Program to demonstrate the MoveChar0 function. }

Var
  Buf1,Buf2 : Array[1..80] of char;
```

```
  I : longint;

begin
  Randomize;
  For I:=low(buf1) to high(buf1) do
    Buf1[i]:=chr(Random(16)+Ord('A'));
  Writeln('Original buffer');
  writeln(Buf1);
  Buf1[Random(80)+1]:=#0;
  MoveChar0(Buf1,Buf2,80);
  Writeln('Randomly zero-terminated Buffer');
  Writeln(Buf2);
end.
```

75.12.229 multiply(variant,variant):variant

Synopsis: Implement multiplication (*) operation on variants.

Declaration: `operator *(const op1: variant; const op2: variant) : variant`

Visibility: default

Description: The implementation of the multiplication * operation is delegated to the variant manager with operation `opMultiply`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator /(variant, variant): variant (1362)

75.12.230 negative(variant):variant

Synopsis: Implement - (unary minus, negation) operation on variants.

Declaration: `operator -(const op: variant) : variant`

Visibility: default

Description: The implementation of the unary minus (-) operation is delegated to the variant manager with operation `varNeg`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator -(variant, variant): variant (1362)

75.12.231 New

Synopsis: Dynamically allocate memory for variable.

Declaration:
```
procedure &New(var P: Pointer)
procedure &New(var P: Pointer; Cons: TProcedure)
```

Visibility: default

Description: `New` allocates a new instance of the type pointed to by `P`, and puts the address in `P`. If P is an object, then it is possible to specify the name of the constructor with which the instance will be created.

The newly allocated memory is not initialized in any way, and may contain garbage data. It must be cleared with a call to FillChar (1493) or FillWord (1494).

For an example, see Dispose (1478).

Errors: What happens if no more memory is available, depends on the value of the variable ReturnNilIfGrowHeapfails (1445): if the variable is `True` then `Nil` is returned. If the variable is `False`, a run-time error is generated.

See also: Dispose (1478), Freemem (1499), Getmem (1502), ReturnNilIfGrowHeapfails (1445)

75.12.232 NtoBE

Synopsis: Convert Native-ordered integer to a Big Endian-ordered integer.

Declaration:
```
function NtoBE(const AValue: SmallInt) : SmallInt
function NtoBE(const AValue: Word) : Word
function NtoBE(const AValue: LongInt) : LongInt
function NtoBE(const AValue: DWord) : DWord
function NtoBE(const AValue: Int64) : Int64
function NtoBE(const AValue: QWord) : QWord
```

Visibility: default

Description: `NToBE` will rearrange the bytes in a natively-ordered number to the Big-Endian order. That is, for a Little-Endian processor, it will invert the order of the bytes and for a big-endian processor, it will do nothing.

See also: BEtoN (1458), LEtoN (1529), NtoLE (1538)

75.12.233 NtoLE

Synopsis: Convert Native-ordered integer to a Little Endian-ordered integer.

Declaration:
```
function NtoLE(const AValue: SmallInt) : SmallInt
function NtoLE(const AValue: Word) : Word
function NtoLE(const AValue: LongInt) : LongInt
function NtoLE(const AValue: DWord) : DWord
function NtoLE(const AValue: Int64) : Int64
function NtoLE(const AValue: QWord) : QWord
```

Visibility: default

Description: `NToLE` will rearrange the bytes in a natively-ordered number to the little-Endian order. That is, for a Big-Endian processor, it will invert the order of the bytes and for a Little-Endian processor, it will do nothing.

See also: BEtoN (1458), LEtoN (1529), NtoBE (1538)

75.12.234 Null

Synopsis: Null variant.

Declaration:
```
function Null : Variant
```

Visibility: default

75.12.235 OctStr

Synopsis: Convert integer to a string with octal representation.

Declaration:
```
function OctStr(Val: LongInt; cnt: Byte) : shortstring
function OctStr(Val: Int64; cnt: Byte) : shortstring
function OctStr(Val: QWord; cnt: Byte) : shortstring
```

Visibility: default

Description: `OctStr` returns a string with the octal representation of `Value`. The string has exactly `cnt` characters.

Errors: None.

See also: Str (1577), Val (1598), BinStr (1458), HexStr (1509)

Listing: ./examples/refex/ex112.pp

```
Program example112;

{ Program to demonstrate the OctStr function }

Const Value = 45678;

Var l : longint;

begin
  For l:=1 to 10 do
    WriteLn (OctStr(Value,l));
  For l:=1 to 16 do
    WriteLn (OctStr(l,3));
end.
```

75.12.236 Odd

Synopsis: Is a value odd or even ?

Declaration:
```
function Odd(l: LongInt) : Boolean
function Odd(l: LongWord) : Boolean
function Odd(l: Int64) : Boolean
function Odd(l: QWord) : Boolean
```

Visibility: default

Description: `Odd` returns `True` if `X` is odd, or `False` otherwise.

Errors: None.

See also: Abs (1446), Ord (1540)

Listing: ./examples/refex/ex43.pp

```
Program Example43;

{ Program to demonstrate the Odd function. }

begin
  If Odd(1) Then
```

```
    WriteLn ('Everything OK with 1 !');
  If Not Odd(2) Then
    WriteLn ('Everything OK with 2 !');
end.
```

75.12.237 Ofs

Synopsis: Return offset of a variable.

Declaration: `function Ofs(var X) : LongInt`

Visibility: default

Description: `Ofs` returns the offset of the address of a variable. This function is only supported for compatibility. In Free Pascal, it returns always the complete address of the variable, since Free Pascal is a 32/64 bit compiler.

Errors: None.

See also: DSeg (1480), CSeg (1475), Seg (1564), Ptr (1545)

Listing: ./examples/refex/ex44.pp

```
Program Example44;

{ Program to demonstrate the Ofs function. }

Var W : Pointer;


begin
  W:=Pointer(Ofs(W)); { W contains its own offset. }
end.
```

75.12.238 Ord

Synopsis: Return ordinal value of an ordinal type.

Declaration: `function Ord(X: TOrdinal) : LongInt`

Visibility: default

Description: `Ord` returns the Ordinal value of a ordinal-type variable `X`.

Historical note:

Originally, Pascal did not have typecasts and ord was a necessary function in order to do certain operations on non-integer ordinal types. With the arrival of typecasting a generic approach became possible, making ord mostly obsolete. However ord is not considered deprecated and remains in wide use today.

Errors: None.

See also: Chr (1465), Succ (1582), Pred (1545), High (1511), Low (1534)

Listing: ./examples/refex/ex45.pp

```
Program Example45;

{ Program to demonstrate the Ord,Pred,Succ functions. }

Type
  TEnum = (Zero, One, Two, Three, Four);

Var
  X : Longint;
  Y : TEnum;

begin
  X:=125;
  Writeln (Ord(X));  { Prints 125 }
  X:=Pred(X);
  Writeln (Ord(X));  { prints 124 }
  Y:= One;
  Writeln (Ord(y));  { Prints 1 }
  Y:=Succ(Y);
  Writeln (Ord(Y));  { Prints 2}
end.
```

75.12.239 Pack

Synopsis: Create packed array from normal array.

Declaration:
```
procedure Pack(const A: UnpackedArrayType; StartIndex: TIndexType;
               out Z: PackedArrayType)
```

Visibility: default

Description: `Pack` will copy the elements of an unpacked array (`A`) to a packed array (`Z`). It will start the copy at the index denoted by `StartIndex`. The type of the index variable `StartIndex` must match the type of the index of `A`. The elements are always transferred to the beginning of the packed array `Z`. (i.e. it starts at `Low(Z)`).

Obviously, the type of the elements of the arrays `A` and `Z` must match.

See also: unpack (1595)

75.12.240 ParamCount

Synopsis: Return number of command-line parameters passed to the program.

Declaration:
```
function ParamCount : LongInt
```

Visibility: default

Description: `Paramcount` returns the number of command-line arguments. If no arguments were given to the running program, `0` is returned.

Errors: None.

See also: Paramstr (1542)

Listing: ./examples/refex/ex46.pp

```
Program Example46;

{ Program to demonstrate the ParamCount and ParamStr functions. }
Var
  I : Longint;

begin
  WriteIn (paramstr(0),' : Got ',ParamCount,' command-line parameters: ');
  For i:=1 to ParamCount do
    WriteIn (ParamStr (i));
end.
```

75.12.241 ParamStr

Synopsis: Return value of a command-line argument.

Declaration: `function ParamStr(l: LongInt) : string`

Visibility: default

Description: `Paramstr` returns the `L`-th command-line argument. `L` must be between `0` and `Paramcount`, these values included. The zeroth argument is the path and file name with which the program was started.

The command-line parameters will be truncated to a length of 255, even though the operating system may support bigger command-lines. The `Objpas` unit (used in `objfpc` or `delphi` mode) defines versions of `Paramstr` which return the full-length command-line arguments, using ansistrings.

In the interest of portability, the `ParamStr` function tries to behave the same on all operating systems: like the original `ParamStr` function in Turbo Pascal. This means even on Unix, paramstr(0) returns the full path to the program executable. A notable exception is Mac OS X, where the returned value depends on how the application was started. It may be that just the name of the application is returned (in case of a command-line launch), so it is best to avoid using it on that platform.

In general, it's a bad idea to rely on the location of the binary. Often, this goes against best OS practices. Configuration data should (or can) not be stored next to the binary, but on designated locations. What locations these are, is very much operating system dependent. Therefore, `ParamStr(0)` should be used with care.

For an example, see Paramcount (1541).

Errors: None.

See also: Paramcount (1541)

75.12.242 Pi

Synopsis: Return the value of PI.

Declaration: `function Pi : ValReal`

Visibility: default

Description: `Pi` returns the value of Pi (3.1415926535897932385).

Errors: None.

See also: Cos (1474), Sin (1573)

Listing: ./examples/refex/ex47.pp

```
Program Example47;

{ Program to demonstrate the Pi function. }

begin
  WriteLn (Pi);          {3.1415926}
  WriteLn (Sin(Pi));
end.
```

75.12.243 PopCnt

Synopsis: Count number of set bits.

Declaration:
```
function PopCnt(const AValue: Byte) : Byte
function PopCnt(const AValue: Word) : Word
function PopCnt(const AValue: DWord) : DWord
function PopCnt(const AValue: QWord) : QWord
```

Visibility: default

Description: `PopCnt` (population count) counts the number of set bits in `AValue`.

75.12.244 Pos

Synopsis: Search for substring in a string.

Declaration:
```
function Pos(const substr: shortstring; const s: shortstring;
             Offset: SizeInt) : SizeInt
function Pos(C: Char; const s: shortstring; Offset: SizeInt) : SizeInt
function Pos(const Substr: ShortString; const Source: RawByteString;
             Offset: SizeInt) : SizeInt
function Pos(const substr: shortstring; c: Char; Offset: SizeInt)
              : SizeInt
function Pos(const Substr: RawByteString; const Source: RawByteString;
             Offset: SizeInt) : SizeInt
function Pos(c: AnsiChar; const s: RawByteString; Offset: SizeInt)
              : SizeInt
function Pos(const Substr: UnicodeString; const Source: UnicodeString;
             Offset: SizeInt) : SizeInt
function Pos(c: Char; const s: UnicodeString; Offset: SizeInt) : SizeInt
function Pos(c: UnicodeChar; const s: UnicodeString; Offset: SizeInt)
              : SizeInt
function Pos(const c: RawByteString; const s: UnicodeString;
             Offset: SizeInt) : SizeInt
function Pos(const c: UnicodeString; const s: RawByteString;
             Offset: SizeInt) : SizeInt
function Pos(const c: ShortString; const s: UnicodeString;
             Offset: SizeInt) : SizeInt
function Pos(const Substr: WideString; const Source: WideString;
             Offset: SizeInt) : SizeInt
function Pos(c: Char; const s: WideString; Offset: SizeInt) : SizeInt
function Pos(c: WideChar; const s: WideString; Offset: SizeInt)
```

```
              : SizeInt
function Pos(c: WideChar; const s: RawByteString; Offset: SizeInt)
              : SizeInt
function Pos(const c: RawByteString; const s: WideString;
             Offset: SizeInt) : SizeInt
function Pos(const c: WideString; const s: RawByteString;
             Offset: SizeInt) : SizeInt
function Pos(const c: ShortString; const s: WideString; Offset: SizeInt)
              : SizeInt
function Pos(c: Char; const v: Variant) : SizeInt
function Pos(s: ShortString; const v: Variant) : SizeInt
function Pos(const a: AnsiString; const v: Variant) : SizeInt
function Pos(const w: WideString; const v: Variant) : SizeInt
function Pos(const w: UnicodeString; const v: Variant) : SizeInt
function Pos(const v: Variant; const c: Char) : SizeInt
function Pos(const v: Variant; const s: ShortString) : SizeInt
function Pos(const v: Variant; const a: AnsiString) : SizeInt
function Pos(const v: Variant; const w: WideString) : SizeInt
function Pos(const v: Variant; const w: UnicodeString) : SizeInt
function Pos(const v1: Variant; const v2: Variant) : SizeInt
```

Visibility: default

Description: `Pos` returns the index of `Substr` (or character `c`)in `S`, if `S` contains `Substr`. In case `Substr` isn't found, `0` is returned. The search is case-sensitive. The character version uses an optimized version of the search algorithm.

In case `Offset` is specified, the search starts at position `Offset` (1-based) in the string; The position `Offset` is also checked in the search.

Errors: None

See also: Length (1528), Copy (1473), Delete (1477), Insert (1521), posEx (1347)

Listing: ./examples/refex/ex48.pp

```
Program Example48;

{ Program to demonstrate the Pos function. }

Var
  S : String;

begin
  S:='The first space in this sentence is at position : ';
  Writeln (S,pos(' ',S));
  S:='The last letter of the alphabet doesn''t appear in this sentence ';
  If (Pos ('Z',S)=0) and (Pos('z',S)=0) then
    Writeln (S);
end.
```

75.12.245 power(variant,variant):variant

Synopsis: Implement power `(**)` operation on variants.

Declaration: `operator **(const op1: variant; const op2: variant) : variant`

Visibility: default

Description: The implementation of the power `**` operation is delegated to the variant manager with operation `opPower`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator *(variant, variant): variant (1362)

75.12.246 Pred

Synopsis: Return previous element for an ordinal type.

Declaration: `function Pred(X: TOrdinal) : TOrdinal`

Visibility: default

Description: `Pred` returns the element that precedes the element that was passed to it. If it is applied to the first value of the ordinal type, and the program was compiled with range checking on (`{$R+}`), then a run-time error will be generated.

For an example, see Ord (1540)

Errors: Run-time error 201 is generated when the result is out of range.

See also: Ord (1540), Succ (1582), High (1511), Low (1534)

75.12.247 Prefetch

Synopsis: Prefetch a memory location.

Declaration: `procedure Prefetch(const mem)`

Visibility: default

Description: `Prefetch` can be used to optimize the CPU behaviour by already loading a memory location. It is mainly used as a hint for those processors that support it.

Errors: None.

75.12.248 Ptr

Synopsis: Combine segment and offset to pointer.

Declaration: `function Ptr(sel: LongInt; off: LongInt) : FarPointer`

Visibility: default

Description: `Ptr` returns a pointer, pointing to the address specified by segment `Sel` and offset `Off`.

Remark

1.In the 32/64-bit flat-memory model supported by Free Pascal, this function is obsolete.

2.The returned address is simply the offset.

Errors: None.

See also: Addr (1448)

Listing: ./examples/refex/ex59.pp

```
Program Example59;

{ Program to demonstrate the Ptr (compatibility) function. }

type pString = ^String;

Var P : pString;
    S : String;

begin
  S:='Hello, World !';
  P:=pString(Ptr(Seg(S),Longint(Ofs(S))));
  {P now points to S !}
  Writeln (P^);
end.
```

75.12.249 RaiseList

Synopsis: List of currently raised exceptions.

Declaration:
```
function RaiseList : PExceptObject
```

Visibility: default

Description: `RaiseList` returns a pointer to the list of currently raised exceptions (i.e. a pointer to the first exception block.

75.12.250 Random

Synopsis: Generate random number.

Declaration:
```
function Random(l: LongInt) : LongInt
function Random(l: Int64) : Int64
function Random : extended
```

Visibility: default

Description: `Random` returns a random number larger or equal to `0` and strictly less than `L`. For negative values of `L` the behaviour is undefined. If the argument `L` is omitted, a Real number between 0 and 1 is returned (0 included, 1 excluded).

Remark The Free Pascal implementation of the `Random` routine uses a Mersenne Twister algorithm to simulate randomness. This implementation has a better statistical distribution than for example a Linear Congruential generator algorithm, but is considerably slower than the latter. If speed is an issue, then alternate random number generators should be considered.

Note that the fact that a Mersenne Twister is used is an implementation detail, which can be changed at any point. The only guarantee `Random()` offers is that setting randseed to particular value will result in the same sequence of random numbers in a particular version the RTL. A newer version of the RTL may result in a different sequence for the same randseed.

Errors: None.

See also: Randomize (1547)

Listing: ./examples/refex/ex49.pp

```
Program Example49;

{ Program to demonstrate the Random and Randomize functions. }

Var I,Count,guess : Longint;
    R : Real;

begin
  Randomize; { This way we generate a new sequence every time
               the program is run}
  Count:=0;
  For i:=1 to 1000 do
    If Random>0.5 then inc(Count);
  Writeln ('Generated ',Count,' numbers > 0.5');
  Writeln ('out of 1000 generated numbers.');
  count:=0;
  For i:=1 to 5 do
    begin
    write ('Guess a number between 1 and 5 : ');
    readln(Guess);
    If Guess=Random(5)+1 then inc(count);
    end;
  Writeln ('You guessed ',Count,' out of 5 correct.');
end.
```

75.12.251 Randomize

Synopsis: Initialize random number generator.

Declaration: `procedure Randomize`

Visibility: default

Description: `Randomize` initializes the random number generator of Free Pascal, by giving a value to `Randseed`, calculated with the system clock.

For an example, see Random (1546).

Errors: None.

See also: Random (1546)

75.12.252 Read

Synopsis: Read from a text file into variable.

Declaration:
```
procedure Read(var F: Text; Args: Arguments)
procedure Read(Args: Arguments)
```

Visibility: default

Description: `Read` reads one or more values from a file `F`, and stores the result in `V1`, `V2`, etc.; If no file `F` is specified, then standard input is read. If `F` is of type `Text`, then the variables `V1, V2` etc. must be of type `Char`, `Integer`, `Real`, `String`. If `F` is a typed file, then each of the variables must be of the type specified in the declaration of `F`. Untyped files are not allowed as an argument.

In earlier versions of FPC, it was also allowed to read `Pchar` null-terminated strings, but this has been removed, since there is no buffer checking possible.

Errors: If no data is available, empty values are returned (0 for ordinal values, empty strings for string values)

See also: Readln (1549), Blockread (1459), Write (1602), Blockwrite (1460)

Listing: ./examples/refex/ex50.pp

```
Program Example50;

{ Program to demonstrate the Read(Ln) function. }

Var S : String;
    C : Char;
    F : File of char;

begin
  Assign (F,'ex50.pp');
  Reset (F);
  C:='A';
  Writeln ('The characters before the first space in ex50.pp are : ');
  While not Eof(f) and (C<>' ') do
    Begin
    Read (F,C);
    Write (C);
    end;
 Writeln;
 Close (F);
 Writeln ('Type some words. An empty line ends the program.');
 repeat
   Readln (S);
 until S='';
end.
```

75.12.253 ReadBarrier

Synopsis: Memory Read Barrier.

Declaration: `procedure ReadBarrier`

Visibility: default

Description: `ReadBarrier` is a low-level instruction to force a read barrier in the CPU: all memory reads before the instruction will be finished before this instruction, before memory reads after the instruction occur.

See also: ReadDependencyBarrier (1548), ReadWriteBarrier (1550), WriteBarrier (1603)

75.12.254 ReadDependencyBarrier

Synopsis: Memory Read Dependency Barrier.

Declaration: `procedure ReadDependencyBarrier`

Visibility: default

Description: `ReadDependencyBarrier` is a low-level instruction to force a read barrier in the CPU: all memory reads (loads) depending on previous loads are separate from the ones following the instruction.

See also: ReadBarrier (1548), ReadWriteBarrier (1550), WriteBarrier (1603)

75.12.255 ReadLn

Synopsis: Read from a text file into variable and goto next line.

Declaration:
```
procedure ReadLn(var F: Text; Args: Arguments)
procedure ReadLn(Args: Arguments)
```

Visibility: default

Description: `Read` reads one or more values from a file `F`, and stores the result in `V1`, `V2`, etc. After that it goes to the next line in the file. The end of the line is marked by any of the supported line ending styles, independent of the platform on which the code is running (supported line ending styles are CRLF, LF or CR). The end-of-line marker is not considered part of the line and is ignored.

If no file `F` is specified, then standard input is read. The variables `V1, V2` etc. must be of type `Char`, `Integer`, `Real`, `String` or `PChar`.

For an example, see Read (1547).

Errors: If no data is available, empty values are returned (0 for ordinal values, empty strings for string values)

See also: Read (1547), Blockread (1459), Write (1602), Blockwrite (1460)

75.12.256 ReadStr

Synopsis: Read variables from a string.

Declaration:
```
procedure ReadStr(const S: string; Args: Arguments)
```

Visibility: default

Description: `ReadStr` behaves like Read (1547), except that it reads its input from the string variable `S` instead of a file. Semantically, the `ReadStr` call is equivalent to writing the string to a file using the `Write` call, and then reading them into the various arguments `Arg` using the `Read` call from the same file:

```
var
  F : Text;
begin
  Rewrite(F);
  Write(F,S);
  Close(F);
  Reset(F);
  Read(F,Args);
  Close(F);
end;
```

Obviously, the `ReadStr` call does not use a temporary file.

`ReadStr` is defined in the ISO Extended Pascal standard. More information on the allowed arguments and the behaviour of the arguments can be found in the description of Read (1547).

See also: Read (1547), WriteStr (1604), Write (1602)

75.12.257 ReadWriteBarrier

Synopsis: Memory read/write barrier.

Declaration: `procedure ReadWriteBarrier`

Visibility: default

Description: `ReadWriteBarrier` is a low-level instruction to force a read/write barrier in the CPU: both read (Loads) and write (stores) operations before and after the barrier are separate.

See also: ReadBarrier (1548), ReadDependencyBarrier (1548), WriteBarrier (1603)

75.12.258 Real2Double

Synopsis: Convert Turbo Pascal style real to double.

Declaration: `function Real2Double(r: Real48) : Double`

Visibility: default

Description: The `Real2Double` function converts a Turbo Pascal style real (6 bytes long) to a native Free Pascal double type. It can be used e.g. to read old binary TP files with FPC and convert them to Free Pascal binary files.

Note that the assignment operator has been overloaded so a `Real48` type can be assigned directly to a double or extended.

Errors: None.

Listing: ./examples/refex/ex110.pp

```
program Example110;

{ Program to demonstrate the Real2Double function. }

Var
  i : integer;
  R : Real48;
  D : Double;
  E : Extended;
  F : File of Real48;

begin
  Assign(F,'reals.dat');
  Reset(f);
  For I:=1 to 10 do
    begin
    Read(F,R);
    D:=Real2Double(R);
    Writeln('Real ',i,' : ',D);
    D:=R;
    Writeln('Real (direct to double)  ',i,' : ',D);
    E:=R;
    Writeln('Real (direct to Extended) ',i,' : ',E);
    end;
  Close(f);
end.
```

75.12.259 ReAllocMem

Synopsis: Re-allocate memory on the heap.

Declaration: `function ReAllocMem(var p: pointer; Size: PtrUInt) : pointer`

Visibility: default

Description: `ReAllocMem` resizes the memory pointed to by `P` so it has size `Size`. The value of `P` may change during this operation. The contents of the memory pointed to by `P` (if any) will be copied to the new location, but may be truncated if the newly allocated memory block is smaller in size. If a larger block is allocated, only the used memory is initialized, extra memory will not be zeroed out.

Note that `P` may be nil, in that case the behaviour of `ReAllocMem` is equivalent to `Getmem`.

See also: GetMem (1502), FreeMem (1499)

75.12.260 ReAllocMemory

Synopsis: Alias for ReAllocMem (1551).

Declaration: `function ReAllocMemory(p: pointer; Size: PtrUInt) : pointer`

Visibility: default

Description: `ReAllocMemory` is an alias for ReAllocMem (1551).

See also: ReAllocMem (1551)

75.12.261 RegisterLazyInitThreadingProc

Declaration: `procedure RegisterLazyInitThreadingProc(const proc: TProcedure)`

Visibility: default

75.12.262 ReleaseExceptionObject

Synopsis: Decrease the reference count of the current exception object.

Declaration: `procedure ReleaseExceptionObject`

Visibility: default

Description: `ReleaseExceptionObject` decreases the reference count of the current exception object. This should be called whenever a reference to the exception object was obtained via the AcquireExceptionObject (1447) call.

Calling this method is only valid within an except block.

Errors: If there is no current exception object, a run-time error 231 will occur.

See also: AcquireExceptionObject (1447)

75.12.263 Rename

Synopsis: Rename file on disk.

Declaration:
```
procedure Rename(var f: File; const s: ShortString)
procedure Rename(var f: File; const p: PAnsiChar)
procedure Rename(var f: File; const c: AnsiChar)
procedure Rename(var f: File; const s: UnicodeString)
procedure Rename(var f: File; const s: RawByteString)
procedure Rename(var t: Text; const s: shortstring)
procedure Rename(var t: Text; const p: PAnsiChar)
procedure Rename(var t: Text; const c: AnsiChar)
procedure Rename(var t: Text; const s: unicodestring)
procedure Rename(var t: Text; const s: RawByteString)
```

Visibility: default

Description: `Rename` changes the name of the assigned file `F` to `S`. `F` must be assigned, but not opened.

Errors: Depending on the state of the `{$I}` switch, a runtime error can be generated if there is an error. In the `{$I-}` state, use `IOResult` to check for errors.

See also: Erase (1486)

Listing: ./examples/refex/ex77.pp

```
Program Example77;

{ Program to demonstrate the Rename function. }
Var F : Text;

begin
  Assign (F,paramstr(1));
  Rename (F,paramstr(2));
end.
```

75.12.264 Reset

Synopsis: Open file for reading.

Declaration:
```
procedure Reset(var f: File; l: LongInt)
procedure Reset(var f: File)
procedure Reset(var f: TypedFile)
procedure Reset(var t: Text)
```

Visibility: default

Description: `Reset` opens a file `F` for reading. `F` can be any file type. If `F` is a text file, or refers to standard I/O (e.g : '') then it is opened read-only, otherwise it is opened using the mode specified in filemode (1372).

If `F` is an untyped file, the record size can be specified in the optional parameter `L`. A default value of 128 is used.

File sharing is not taken into account when calling `Reset`.

Note that the path can be only 255 characters long.

Errors: Depending on the state of the `{$I}` switch, a runtime error can be generated if there is an error. In the `{$I-}` state, use `IOResult` to check for errors.

See also: Rewrite (1553), Assign (1451), Close (1465), Append (1449), FileMode (1372)

Listing: ./examples/refex/ex51.pp

```
Program Example51;

{ Program to demonstrate the Reset function. }

Function FileExists (Name : String) : boolean;

Var F : File;

begin
  {$i-}
  Assign (F,Name);
  Reset (F);
  {$I+}
  FileExists:=(IoResult=0) and (Name<>'');
  if FileExists then
    Close (f);
end;

begin
  If FileExists (Paramstr(1)) then
    WriteIn ('File found')
  else
    WriteIn ('File NOT found');
end.
```

75.12.265 ResumeThread

Synopsis: Resume a suspended thread.

Declaration: `function ResumeThread(threadHandle: TThreadID) : DWord`

Visibility: default

Description: `ResumeThread` causes a suspended thread (using SuspendThread (1583)) to resume it's execution. The thread is identified with it's handle or ID `threadHandle`.

The function returns zero if successful. A nonzero return value indicates failure.

Errors: If a failure occurred, a nonzero result is returned. The meaning is system dependent.

See also: SuspendThread (1583), KillThread (1527)

75.12.266 Rewrite

Synopsis: Open file for writing.

Declaration:
```
procedure Rewrite(var f: File; l: LongInt)
procedure Rewrite(var f: File)
procedure Rewrite(var f: TypedFile)
procedure Rewrite(var t: Text)
```

Visibility: default

Description: `Rewrite` opens a file `F` for writing. `F` can be any file type. If `F` is an untyped or typed file, then it is opened for reading and writing. If `F` is an untyped file, the record size can be specified in the optional parameter `L`. Default a value of 128 is used. if `Rewrite` finds a file with the same name as `F`, this file is truncated to length `0`. If it doesn't find such a file, a new file is created.

Contrary to Turbo Pascal, Free Pascal opens the file with mode `fmoutput`. If it should be opened in `fminout` mode, an extra call to Reset (1552) is needed.

File sharing is not taken into account when calling `Rewrite`.

Note that the path can be only 255 characters long.

Errors: Depending on the state of the `{$I}` switch, a runtime error can be generated if there is an error. In the `{$I-}` state, use `IOResult` to check for errors.

See also: Reset (1552), Assign (1451), Close (1465), Flush (1497), Append (1449)

Listing: ./examples/refex/ex52.pp

```
Program Example52;

{ Program to demonstrate the Rewrite function. }

Var F : File;
    I : longint;

begin
  Assign (F,'Test.tmp');
  { Create the file. Recordsize is 4 }
  Rewrite (F,Sizeof(I));
  For I:=1 to 10 do
    BlockWrite (F,I,1);
  close (f);
  { F contains now a binary representation of
    10 longints going from 1 to 10 }
end.
```

75.12.267 rightshift(variant,variant):variant

Synopsis: Implement binary `shr` operation on variants.

Declaration: `operator shr(const op1: variant; const op2: variant) : variant`

Visibility: default

Description: The implementation of the `shr` operation is delegated to the variant manager with operation `opshiftright`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator shl(variant, variant): variant (1362)

75.12.268 RmDir

Synopsis: Remove directory when empty.

Declaration:
```
procedure RmDir(const s: shortstring);  Overload
procedure RmDir(const s: RawByteString);  Overload
procedure RmDir(const s: unicodestring);  Overload
```

Visibility: default

Description: `Rmdir` removes the directory `S`.

Errors: Depending on the state of the `{$I}` switch, a runtime error can be generated if there is an error. In the `{$I-}` state, use `IOResult` to check for errors.

See also: Chdir (1464), Mkdir (1535)

Listing: ./examples/refex/ex53.pp

```
Program Example53;

{ Program to demonstrate the MkDir and RmDir functions. }

Const D : String[8] = 'TEST.DIR';

Var S : String;

begin
  WriteLn ('Making directory ',D);
  Mkdir (D);
  WriteLn ('Changing directory to ',D);
  ChDir (D);
  GetDir (0,S);
  WriteLn ('Current Directory is : ',S);
  WRiteln ('Going back');
  ChDir ('..');
  WriteLn ('Removing directory ',D);
  RmDir (D);
end.
```

75.12.269 RolByte

Synopsis: Rotate bits of a byte value to the left.

Declaration:
```
function RolByte(const AValue: Byte) : Byte
function RolByte(const AValue: Byte; const Dist: Byte) : Byte
```

Visibility: default

Description: `RolByte` rotates the bits of the byte `AValue` with `Dist` positions to the left. If `Dist` is not specified, then 1 is assumed.

Errors: None.

See also: RorByte (1556), RolWord (1556), RolDWord (1555), RolQWord (1556)

75.12.270 RolDWord

Synopsis: Rotate bits of a DWord (cardinal) value to the left.

Declaration:
```
function RolDWord(const AValue: DWord) : DWord
function RolDWord(const AValue: DWord; const Dist: Byte) : DWord
```

Visibility: default

Description: `RolDWord` rotates the bits of the DWord (cardinal) `AValue` with `Dist` positions to the left. If `Dist` is not specified, then 1 is assumed.

Errors: None.

See also: RolByte (1555), RolWord (1556), RorDWord (1557), RolQWord (1556)

75.12.271 RolQWord

Synopsis: Rotate bits of a QWord (64-bit) value to the left.

Declaration:
```
function RolQWord(const AValue: QWord) : QWord
function RolQWord(const AValue: QWord; const Dist: Byte) : QWord
```

Visibility: default

Description: `RorQWord` rotates the bits of the QWord (64-bit) `AValue` with `Dist` positions to the left. If `Dist` is not specified, then 1 is assumed.

Errors: None.

See also: RolByte (1555), RolWord (1556), RolDWord (1555), RorQWord (1557)

75.12.272 RolWord

Synopsis: Rotate bits of a word value to the left.

Declaration:
```
function RolWord(const AValue: Word) : Word
function RolWord(const AValue: Word; const Dist: Byte) : Word
```

Visibility: default

Description: `RolWord` rotates the bits of the word `AValue` with `Dist` positions to the right. If `Dist` is not specified, then 1 is assumed.

Errors: None.

See also: RolByte (1555), RorWord (1557), RolDWord (1555), RolQWord (1556)

75.12.273 RorByte

Synopsis: Rotate bits of a byte value to the right.

Declaration:
```
function RorByte(const AValue: Byte) : Byte
function RorByte(const AValue: Byte; const Dist: Byte) : Byte
```

Visibility: default

Description: `RorByte` rotates the bits of the byte `AValue` with `Dist` positions to the right. If `Dist` is not specified, then 1 is assumed.

Errors: None.

See also: RolByte (1555), RorWord (1557), RorDWord (1557), RorQWord (1557)

75.12.274 RorDWord

Synopsis: Rotate bits of a DWord (cardinal) value to the right.

Declaration:
```
function RorDWord(const AValue: DWord) : DWord
function RorDWord(const AValue: DWord; const Dist: Byte) : DWord
```

Visibility: default

Description: `RorDWord` rotates the bits of the DWord (cardinal) `AValue` with `Dist` positions to the right. If `Dist` is not specified, then 1 is assumed.

Errors: None.

See also: RorByte (1556), RolDWord (1555), RorWord (1557), RorQWord (1557)

75.12.275 RorQWord

Synopsis: Rotate bits of a QWord (64-bit) value to the right.

Declaration:
```
function RorQWord(const AValue: QWord) : QWord
function RorQWord(const AValue: QWord; const Dist: Byte) : QWord
```

Visibility: default

Description: `RorQWord` rotates the bits of the QWord (64-bit) `AValue` with `Dist` positions to the right. If `Dist` is not specified, then 1 is assumed.

Errors: None.

See also: RorByte (1556), RorWord (1557), RorDWord (1557), RolQWord (1556)

75.12.276 RorWord

Synopsis: Rotate bits of a word value to the right.

Declaration:
```
function RorWord(const AValue: Word) : Word
function RorWord(const AValue: Word; const Dist: Byte) : Word
```

Visibility: default

Description: `RorWord` rotates the bits of the word `AValue` with `Dist` positions to the right. If `Dist` is not specified, then 1 is assumed.

Errors: None.

See also: RorByte (1556), RolWord (1556), RorDWord (1557), RorQWord (1557)

75.12.277 Round

Synopsis: Round floating point value to nearest integer number.

Declaration:
```
function Round(d: ValReal) : Int64
```

Visibility: default

Description: `Round` rounds `X` to the closest integer, which may be bigger or smaller than `X`.

In the case of `.5`, the algorithm uses "banker's rounding": `.5` values are always rounded towards the even number.

Errors: An invalid floating point operation error may occur in one of several conditions:

- `Infinity` or `NegInfinity` is specified.
- `Nan` is specified.
- `X` is out of the valid range of integers.

See also: Frac (1499), Int (1521), Trunc (1589)

Listing: ./examples/refex/ex54.pp

```
Program Example54;

{ Program to demonstrate the Round function. }

begin
  Writeln (Round(1234.56)); { Prints 1235     }
  Writeln (Round(-1234.56)); { Prints -1235    }
  Writeln (Round(12.3456)); { Prints 12       }
  Writeln (Round(-12.3456)); { Prints -12      }
  Writeln (Round(2.5));      { Prints 2 (down) }
  Writeln (Round(3.5));      { Prints 4 (up)   }

end.
```

Listing: ./examples/refex/ex125.pp

```
program ex124;

{
  Example to show various error conditions for the Round() function.
  SysUtils is used to convert the runtime error to an exception that can be caught
  Math is used to have access to Nan and (Neg)Infinity

}

{$mode objfpc}
{$h+}
uses

  sysutils, math;
var
  D: Double;
  Q: QWord;
  L: Int64;
begin
  D := 2.0 * High(QWord);
  try
    Q := Round(D);
  except
    on E: Exception do writeln(E.ClassName,': ',E.Message);   //EInvalidOp: Invalid floating p
  end;
  D := NaN;
  try
    Q := Round(D);
  except
    on E: Exception do writeln(E.ClassName,': ',E.Message);   //EInvalidOp: Invalid floating p
  end;
  D := Infinity;
```

```
  try
    Q := Round(D);
  except
    on E: Exception do writeln(E.ClassName,': ',E.Message);   //EInvalidOp: Invalid floating p
  end;
  D := NegInfinity;
  try
    Q := Round(D);
  except
    on E: Exception do writeln(E.ClassName,': ',E.Message);   //EInvalidOp: Invalid floating p
  end;
  D := -2.0 * High(QWord);
  try
    L := Round(D);
  except
    on E: Exception do writeln(E.ClassName,': ',E.Message);   //EInvalidOp: Invalid floating p
  end;
end.
```

75.12.278 RTLEventCreate

Synopsis: Create a new RTL event.

Declaration: `function RTLEventCreate : PRTLEvent`

Visibility: default

Description: `RTLEventCreate` creates and initializes a new RTL event. RTL events are used to notify other threads that a certain condition is met, and to notify other threads of condition changes (conditional variables).

The function returns an initialized RTL event, which must be disposed of with RTLEventdestroy (1559)

`RTLEvent` is used mainly for the synchronize method.

See also: RTLEventDestroy (1559), RTLEventSetEvent (1560), RTLEventReSetEvent (1559), RTLEvent-WaitFor (1560)

75.12.279 RTLEventDestroy

Synopsis: Destroy a RTL Event.

Declaration: `procedure RTLEventDestroy(state: PRTLEvent)`

Visibility: default

Description: `RTLeventdestroy` destroys the RTL event `State`. After a call to `RTLeventdestroy`, the `State` RTL event may no longer be used.

See also: RTLEventCreate (1559), RTLEventResetEvent (1559), RTLEventSetEvent (1560)

75.12.280 RTLEventResetEvent

Synopsis: Reset an event.

Declaration: `procedure RTLEventResetEvent(state: PRTLEvent)`

Visibility: default

Description: `RTLeventResetEvent` resets the event: this should be used to undo the signaled state of an event. Resetting an event that is not set (or was already reset) has no effect.

See also: RTLEventCreate (1559), RTLEventDestroy (1559), RTLEventSetEvent (1560), RTLEventWaitFor (1560)

75.12.281 RTLEventSetEvent

Synopsis: Notify threads of the event.

Declaration:
```
procedure RTLEventSetEvent(state: PRTLEvent)
```

Visibility: default

Description: `RTLeventSetEvent` notifies other threads which are listening, that the event has occurred.

See also: RTLEventCreate (1559), RTLEventResetEvent (1559), RTLEventDestroy (1559), RTLEventWaitFor (1560)

75.12.282 RTLEventWaitFor

Synopsis: Wait for an event.

Declaration:
```
procedure RTLEventWaitFor(state: PRTLEvent)
procedure RTLEventWaitFor(state: PRTLEvent; timeout: LongInt)
```

Visibility: default

Description: `RTLeventWaitFor` suspends the thread till the event occurs. The event will occur when another thread calls RTLEventSetEvent (1560) on `State`.

By default, the thread will be suspended indefinitely. However, if `TimeOut` is specified, then the thread will resume after timeout milliseconds have elapsed.

See also: RTLEventCreate (1559), RTLEventDestroy (1559), RTLEventSetEvent (1560), RTLEventWaitFor (1560)

75.12.283 RunError

Synopsis: Generate a run-time error.

Declaration:
```
procedure RunError(w: Word)
procedure RunError
```

Visibility: default

Description: `Runerror` stops the execution of the program, and generates a run-time error `ErrorCode`.

Errors: None.

See also: Exit (1488), Halt (1509)

Listing: ./examples/refex/ex55.pp

```
Program Example55;

{ Program to demonstrate the RunError function. }

begin
  { The program will stop end emit a run-error 106 }
  RunError (106);
end.
```

75.12.284 SafeLoadLibrary

Synopsis: Load a library safely.

Declaration:
```
function SafeLoadLibrary(const Name: RawByteString) : TLibHandle
function SafeLoadLibrary(const Name: UnicodeString) : TLibHandle
```

Visibility: default

Description: `SafeLoadLibrary` calls LoadLibrary (1531) but restores the current FPU control word and exception mask to their current value after the library was loaded, thus preventing the loaded library initialization code from modifying their current values.

See also: LoadLibrary (1531)

75.12.285 SarInt64

Synopsis: 64-bit Shift Arithmetic Right.

Declaration:
```
function SarInt64(const AValue: Int64) : Int64
function SarInt64(const AValue: Int64; Shift: Byte) : Int64
```

Visibility: default

Description: `SarInt64` performs an arithmetic right shift for `Shift` positions on a 64-bit integer `AValue` and returns the result. `Shift` is optional, and is 1 by default. The difference with the regular `Shr` shift operation is that the leftmost bit is preserved during the shift operation.

See also: SarShortInt (1562), SarSmallInt (1562), SarLongInt (1561)

75.12.286 SarLongint

Synopsis: 32-bit Shift Arithmetic Right.

Declaration:
```
function SarLongint(const AValue: LongInt; const Shift: Byte) : LongInt
```

Visibility: default

Description: `SarLongint` performs an arithmetic right shift for `Shift` positions on a 32-bit integer `AValue` and returns the result. `Shift` is optional, and is 1 by default. The difference with the regular `Shr` shift operation is that the leftmost bit is preserved during the shift operation.

See also: SarShortInt (1562), SarSmallInt (1562), SarInt64 (1561)

75.12.287 SarShortint

Synopsis: 8-bit Shift Arithmetic Right.

Declaration:
```
function SarShortint(const AValue: ShortInt; const Shift: Byte)
                            : ShortInt
```

Visibility: default

Description: `SarShortint` performs an arithmetic right shift for `Shift` positions on an 8-bit integer `AValue` and returns the result. `Shift` is optional, and is 1 by default. The difference with the regular `Shr` shift operation is that the leftmost bit is preserved during the shift operation.

See also: SarSmallint (1562), SarLongint (1561), SarInt64 (1561)

75.12.288 SarSmallint

Synopsis: 16-bit Shift Arithmetic Right.

Declaration:
```
function SarSmallint(const AValue: SmallInt; const Shift: Byte)
                            : SmallInt
```

Visibility: default

Description: `SarSmallint` performs an arithmetic right shift for `Shift` positions on an 16-bit integer `AValue` and returns the result. `Shift` is optional, and is 1 by default. The difference with the regular `Shr` shift operation is that the leftmost bit is preserved during the shift operation.

See also: SarShortint (1562), SarLongint (1561), SarInt64 (1561)

75.12.289 Seek

Synopsis: Set file position.

Declaration:
```
procedure Seek(var f: File; Pos: Int64)
```

Visibility: default

Description: `Seek` sets the file-pointer for file `F` to record Nr. `Count`. The first record in a file has `Count=0`. F can be any file type, except `Text`. If `F` is an untyped file, with no record size specified in Reset (1552) or Rewrite (1553), 128 is assumed.

Errors: Depending on the state of the `{$I}` switch, a runtime error can be generated if there is an error. In the `{$I-}` state, use `IOResult` to check for errors.

See also: Eof (1484), SeekEof (1563), SeekEoln (1564)

Listing: ./examples/refex/ex56.pp

```
Program Example56;

{ Program to demonstrate the Seek function. }

Var
  F : File;
  I,j : longint;

begin
  { Create a file and fill it with data }
```

```
  Assign (F,'test.tmp');
  Rewrite(F); { Create file }
  Close(f);
  FileMode:=2;
  ReSet (F,Sizeof(i)); { Opened read/write }
  For I:=0 to 10 do
    BlockWrite (F,I,1);
  { Go Back to the begining of the file }
  Seek(F,0);
  For I:=0 to 10 do
    begin
    BlockRead (F,J,1);
    If J<>I then
      WriteIn ('Error: expected ' ,i,', got ',j);
    end;
  Close (f);
end.
```

75.12.290 SeekEOF

Synopsis: Set file position to end of file.

Declaration:
```
function SeekEOF(var t: Text) : Boolean
function SeekEOF : Boolean
```

Visibility: default

Description: `SeekEof` returns `True` is the file-pointer is at the end of the file. It ignores all whitespace. Calling this function has the effect that the file-position is advanced until the first non-whitespace character or the end-of-file marker is reached.

If the end-of-file marker is reached, `True` is returned. Otherwise, False is returned.

If the parameter `F` is omitted, standard `Input` is assumed.

Remark The SeekEOF function can only be used on real textfiles: when assigning the file to other kinds of (virtual) text files, the function may fail, although it will perform a number of tests to guard against wrong usage.

Errors: A run-time error is generated if the file `F` isn't opened.

See also: Eof (1484), SeekEoln (1564), Seek (1562)

Listing: ./examples/refex/ex57.pp

```
Program Example57;

{ Program to demonstrate the SeekEof function. }
Var C : Char;

begin
  { this will print all characters from standard input except
    Whitespace characters. }
  While Not SeekEof do
    begin
    Read (C);
    Write (C);
    end;
end.
```

75.12.291 SeekEOLn

Synopsis: Set file position to end of line.

Declaration:
```
function SeekEOLn(var t: Text) : Boolean
function SeekEOLn : Boolean
```

Visibility: default

Description: `SeekEoln` returns `True` is the file-pointer is at the end of the current line. It ignores all whitespace. Calling this function has the effect that the file-position is advanced until the first non-whitespace character or the end-of-line marker is reached. If the end-of-line marker is reached, `True` is returned. Otherwise, False is returned. The end-of-line marker is defined as `#10`, the LineFeed character. If the parameter `F` is omitted, standard `Input` is assumed.

Errors: A run-time error is generated if the file `F` isn't opened.

See also: Eof (1484), SeekEof (1563), Seek (1562)

Listing: ./examples/refex/ex58.pp

```
Program Example58;

{ Program to demonstrate the SeekEoln function. }
Var
  C : Char;

begin
  { This will read the first line of standard output and print
    all characters except whitespace. }
  While not SeekEoln do
    Begin
    Read (c);
    Write (c);
    end;
end.
```

75.12.292 Seg

Synopsis: Return segment.

Declaration:
```
function Seg(var X) : LongInt
```

Visibility: default

Description: `Seg` returns the segment of the address of a variable. This function is only supported for compatibility. In Free Pascal, it returns always 0, since Free Pascal uses a flat 32/64 bit memory model. In such a memory model segments have no meaning.

Errors: None.

See also: DSeg (1480), CSeg (1475), Ofs (1540), Ptr (1545)

Listing: ./examples/refex/ex60.pp

```
Program Example60;

{ Program to demonstrate the Seg function. }
Var
```

```
  W : Word;

begin
  W:=Seg(W);   { W contains its own Segment}
end.
```

75.12.293 Set8087CW

Declaration: `procedure Set8087CW(cw: Word)`

Visibility: default

75.12.294 SetCodePage

Synopsis: Set the codepage of a string.

Declaration:
```
procedure SetCodePage(var s: RawByteString; CodePage: TSystemCodePage;
                      Convert: Boolean)
```

Visibility: default

Description: `SetCodePage` sets the codepage of a string `S` to `CodePage`. If `Convert` is `True` then the string will be transcoded to the new codepage. The resulting string will have reference count 1.

See also: StringCodePage (1579)

75.12.295 SetDynLibsManager

Synopsis: Set a new handler for dynamic library support.

Declaration:
```
procedure SetDynLibsManager(const &New: TDynLibsManager)
procedure SetDynLibsManager(const &New: TDynLibsManager;
                            var Old: TDynLibsManager)
```

Visibility: default

Description: `SetDynLibsManager` sets the handler for dynamic library support to `New`. Optionally, it returns the currently active handler in `Old`.

This is the routine used by the #rtl.dynlibs (734) unit to set the default handler.

See also: #rtl.dynlibs (734), GetDynLibsManager (1502)

75.12.296 Setjmp

Synopsis: Save current execution point.

Declaration: `function Setjmp(var S: jmp_buf) : LongInt`

Visibility: default

Description: `SetJmp` fills `S` with the necessary data for a jump back to the point where it was called. It returns zero if called in this way. If the function returns nonzero, then it means that a call to LongJmp (1533) with `S` as an argument was made somewhere in the program.

Errors: None.

See also: LongJmp (1533)

Listing: ./examples/refex/ex79.pp

```
program example79;

{ Program to demonstrate the setjmp, longjmp functions }

procedure dojmp(var env : jmp_buf; value : longint);

begin
  value:=2;
  WriteIn ('Going to jump !');
  { This will return to the setjmp call,
    and return value instead of 0 }
  longjmp(env,value);
end;

var env : jmp_buf;

begin
  if setjmp(env)=0 then
    begin
    writeln ('Passed first time.');
    dojmp(env,2);
    end
  else
    writeln ('Passed second time.');
end.
```

75.12.297 SetLength

Synopsis: Set length of a string or dynamic array.

Declaration:
```
procedure &SetLength(var S: AStringType; Len: SizeInt)
procedure &SetLength(var A: DynArrayType; Len: SizeInt)
```

Visibility: default

Description: `SetLength` for strings sets the length of the string `S` to `Len`. `S` can be an ansistring, a short string or a widestring. For `ShortStrings`, `Len` can maximally be 255. For `AnsiStrings` it can have any value. For `AnsiString` strings, `SetLength`**must** be used to set the length of the string.

In the case of a dynamic array `A`, `SetLength` sets the number of elements. The elements are numbered from index 0, so the count runs from 0 to `Len-1`. If Zero is specified, the array is cleared and removed from memory.

In case the length is set to a smaller length than the current one, the elements `0-(Len-1)` (or characters `1-Len` in case of a string) are kept. The elements that fall outside the new length are finalized if the array element is a managed type.

In case the length is set to a larger length than the current one, the new elements are zeroed out for a dynamic array. For a string, the string is zero-terminated at the correct length.

Note that `SetLength` is governed by the Copy-On-Write principle for strings and dynamic arrays: the reference count after a call to `SetLength` will be 1 (except when the length is zero, then the array is removed from memory).

For multi-dimensional arrays, `SetLength` can be used to specify all dimensions at once:

```
Var
  arr : Array of Array of Integer;

begin
  SetLength(arr, 10, 20);
end;
```

This will create a dynamic array with 10 elements, where each element in itself is a dynamic array of 20 elements. `SetLength` will of course always create "rectangular" arrays: all elements will have the same size.

```
Var
  arr : Array of Integer;

begin
  SetLength(arr,0);
end;
```

After this, `arr` is `Nil`.

Errors: None.

See also: Length (1528)

Listing: ./examples/refex/ex85.pp

```
Program Example85;

{ Program to demonstrate the SetLength function. }

Var S : String;

begin
  Setlength(S,100);
  FillChar(S[1],100,#32);
  Writeln ('"',S,'"');
end.
```

75.12.298 SetMemoryManager

Synopsis: Set a memory manager.

Declaration: `procedure SetMemoryManager(const MemMgr: TMemoryManager)`

Visibility: default

Description: `SetMemoryManager` sets the current memory manager record to `MemMgr`.

For an example, see the programmer's guide.

Errors: None.

See also: GetMemoryManager (1503), IsMemoryManagerSet (1527)

75.12.299 SetMultiByteConversionCodePage

Synopsis: Set codepage for conversions from multi-byte strings to single-byte strings.

Declaration: `procedure SetMultiByteConversionCodePage(CodePage: TSystemCodePage)`

Visibility: default

Description: `SetMultiByteConversionCodePage` sets DefaultSystemCodePage (1443) to `CodePage`. The effect of this change is that the default codepage used to translate multi-byte (UTF-16) strings to single-byte codepage-aware strings changes, and code page conversions will be done to the new codepage.

Do not set `DefaultSystemCodePage` directly, as additional actions may need to be done when changing the code page.

When the DefaultSystemCodePage (1443) changes and you use the `TEncoding` classes, you must call TEncoding.FreeEncodings (1869) to regenerate the default encoding using the new code page.

See also: DefaultSystemCodePage (1443), SetMultiByteFileSystemCodePage (1568), SetMultiByteRTLFileSystemCodePage (1568)

75.12.300 SetMultiByteFileSystemCodePage

Synopsis: Set codepage used when passing strings to OS single-byte file system APIs.

Declaration: `procedure SetMultiByteFileSystemCodePage(CodePage: TSystemCodePage)`

Visibility: default

Description: `SetMultiByteFileSystemCodePage` sets the codepage used in single-byte OS file system APIs to `CodePage`. The effect of this change is that the default codepage used to translate multi-byte (UTF-16) strings to single-byte codepage-aware strings used in File system APIs changes, and strings passed to the codepage-aware file system APIs will be passed using the new codepage.

This constant is not used if the file system API of the OS is multi-byte (such as on Windows).

Do not set `DefaultFileSystemCodePage` directly, as additional actions may need to be done when changing the code page.

See also: DefaultFileSystemCodePage (1442), SetMultiByteConversionCodePage (1568), SetMultiByteRTLFileSystemCodePage (1568)

75.12.301 SetMultiByteRTLFileSystemCodePage

Synopsis: Set codepage used when interpreting strings from OS single-byte file system APIs.

Declaration: `procedure SetMultiByteRTLFileSystemCodePage(CodePage: TSystemCodePage)`

Visibility: default

Description: `SetMultiByteRTLFileSystemCodePage` sets the codepage used to interpreted strings returned by single-byte OS file system APIs to `CodePage`.

The effect of this change is that the default codepage used to translate single byte strings obtained from the OS to single-byte codepage-aware strings or multi-byte strings changes, and strings obtained from the codepage-aware file system APIs will be interpreted using the new codepage.

his constant is not used if the file system API of the OS is multi-byte (such as on Windows).

Do not set `DefaultRTLFileSystemCodePage` directly, as additional actions may need to be done when changing the code page.

See also: SetMultiByteFileSystemCodePage (1568), SetMultiByteConversionCodePage (1568), SetMultiByteRTL-FileSystemCodePage (1568)

75.12.302 SetMXCSR

Declaration:
```
procedure SetMXCSR(w: DWord)
```

Visibility: default

75.12.303 SetResourceManager

Synopsis: Set the resource manager.

Declaration:
```
procedure SetResourceManager(const &New: TResourceManager)
```

Visibility: default

Description: `SetResourceManager` sets the active resource manager to `Manager`. After a call to `SetResourceManager`, the functions in the `Manager` record will be used to handle resources.

Note that it is not supported to change resource managers on-the-fly: any resources or information about resources obtained should be discarded prior to a call to `SetResourceManager`. Typically, `SetResourceManager` should be called once, at program startup.

Errors: None.

See also: TResourceManager (1423), GetResourceManager (1504)

75.12.304 SetSSECSR

Declaration:
```
procedure SetSSECSR(w: DWord)
```

Visibility: default

75.12.305 SetString

Synopsis: Set length of a string and copy buffer.

Declaration:
```
procedure SetString(out S: AnsiString; Buf: PAnsiChar; Len: SizeInt)
procedure SetString(out S: AnsiString; Buf: PWideChar; Len: SizeInt)
procedure SetString(out S: Shortstring; Buf: PChar; Len: SizeInt)
procedure SetString(out S: UnicodeString; Buf: PUnicodeChar;
                    Len: SizeInt)
procedure SetString(out S: UnicodeString; Buf: PChar; Len: SizeInt)
procedure SetString(out S: WideString; Buf: PWideChar; Len: SizeInt)
procedure SetString(out S: WideString; Buf: PChar; Len: SizeInt)
```

Visibility: default

Description: `SetString` sets the length of the string `S` to `Len` and if `Buf` is non-nil, copies `Len` characters from `Buf` into `S`. `S` can be an ansistring, a short string or a widestring. For `ShortStrings`, `Len` can maximally be 255.

Errors: None.

See also: SetLength (1566)

75.12.306 SetTextBuf

Synopsis: Set size of text file internal buffer.

Declaration:
```
procedure SetTextBuf(var f: Text; var Buf)
procedure SetTextBuf(var f: Text; var Buf; Size: SizeInt)
```

Visibility: default

Description: `SetTextBuf` assigns an I/O buffer to a text file. The new buffer is located at `Buf` and is `Size` bytes long. If `Size` is omitted, then `SizeOf(Buf)` is assumed. The standard buffer of any text file is 128 bytes long. For heavy I/O operations this may prove too slow. The `SetTextBuf` procedure allows to set a bigger buffer for the I/O of the application, thus reducing the number of system calls, and thus reducing the load on the system resources. The maximum size of the newly assigned buffer is 65355 bytes.

Remark

- Never assign a new buffer to an opened file. A new buffer can be assigned immediately after a call to Rewrite (1553), Reset (1552) or `Append`, but not after the file was read from/written to. This may cause loss of data. If a new buffer must be assigned after read/write operations have been performed, the file should be flushed first. This will ensure that the current buffer is emptied.
- Take care that the assigned buffer is always valid. If a local variable is assigned as a buffer, then after the program exits the local program block, the buffer will no longer be valid, and stack problems may occur.

Errors: No checking on `Size` is done.

See also: Assign (1451), Reset (1552), Rewrite (1553), Append (1449)

Listing: ./examples/refex/ex61.pp

```
Program Example61;

{ Program to demonstrate the SetTextBuf function. }

Var
  Fin,Fout : Text;
  Ch : Char;
  Bufin,Bufout : Array[1..10000] of byte;

begin
  Assign (Fin,paramstr(1));
  Reset (Fin);
  Assign (Fout,paramstr(2));
  Rewrite (Fout);
  { This is harmless before IO has begun }
  { Try this program again on a big file,
    after commenting out the following 2
    lines and recompiling it. }
  SetTextBuf (Fin,Bufin);
  SetTextBuf (Fout,Bufout);
  While not eof(Fin) do
    begin
    Read (Fin,ch);
    write (Fout,ch);
    end;
  Close (Fin);
```

```
  Close (Fout);
end.
```

75.12.307 SetTextCodePage

Synopsis: Set the codepage used in a text file.

Declaration: `procedure SetTextCodePage(var T: Text; CodePage: TSystemCodePage)`

Visibility: default

Description: `GetTextCodePage` sets the codepage that the text file `T` uses. All strings written to the file will be converted to the indicated codepage. By default, the codepage is set to `CP_ACP`.

Errors: None.

See also: TextRec (1416), GetTextCodePage (1505)

75.12.308 SetTextLineEnding

Synopsis: Set the end-of-line character for the given text file.

Declaration: `procedure SetTextLineEnding(var f: Text; Ending: string)`

Visibility: default

Description: `SetTextLineEnding` sets the end-of-line character for the text file `F` to `Ending`. By default, this is the string indicated by DefaultTextLineBreakStyle (1371).

Errors: None.

See also: DefaultTextLineBreakStyle (1371), TTextLineBreakStyle (1427)

75.12.309 SetThreadDebugName

Synopsis: Set the debug name for a thread.

Declaration:
```
procedure SetThreadDebugName(threadHandle: TThreadID;
                             const ThreadName: AnsiString)
procedure SetThreadDebugName(threadHandle: TThreadID;
                             const ThreadName: UnicodeString)
```

Visibility: default

Description: `SetThreadDebugName` can be used to set the name of a thread in a debugging context. A debugger can use this information to show the name instead of the thread ID, thus making debugging easier. The current platform's TThreadManager (1428) implementation must support this call.

Errors: This call does nothing on platforms that do not support threading.

See also: GetCurrentThreadID (1501), TThreadManager (1428)

75.12.310 SetThreadManager

Synopsis: Set the thread manager, optionally return the current thread manager.

Declaration:
```
function SetThreadManager(const NewTM: TThreadManager;
                          var OldTM: TThreadManager) : Boolean
function SetThreadManager(const NewTM: TThreadManager) : Boolean
```

Visibility: default

Description: `SetThreadManager` sets the thread manager to `NewTM`. If `OldTM` is given, `SetThreadManager` uses it to return the previously used thread manager.

The function returns `True` if the threadmanager was set successfully, `False` if an error occurred.

For more information about thread programming, see the programmer's guide.

Errors: If an error occurred cleaning up the previous manager, or an error occurred initializing the new manager, `False` is returned.

See also: GetThreadManager (1505), TThreadManager (1428)

75.12.311 SetUnicodeStringManager

Synopsis: Set the unicodestring manager.

Declaration:
```
procedure SetUnicodeStringManager(const &New: TUnicodeStringManager)
procedure SetUnicodeStringManager(const &New: TUnicodeStringManager;
                                  var Old: TUnicodeStringManager)
```

Visibility: default

Description: `SetUnicodeStringManager` sets the current unicodestring manager to `New`. Optionally, it returns the currently active widestring manager in `Old`.

UnicodeStrings are implemented in different ways on different platforms. Therefore, the Free Pascal Runtime library has no fixed implementation of widestring routines. Instead, it defines a UnicodeString manager record, with callbacks that can be set to an implementation which is most efficient on the current platform. On windows, standard Windows routines will be used. On Unix and Linux, an implementation based on the C library is available (in unit cwstring).

It is possible to implement a custom unicodestring manager, optimized for the current application, without having to recompile the complete Run-Time Library.

See also: TUnicodeStringManager (1431)

75.12.312 SetVariantManager

Synopsis: Set the current variant manager.

Declaration:
```
procedure SetVariantManager(const VarMgr: tvariantmanager)
```

Visibility: default

Description: `SetVariantManager` sets the variant manager to `varmgr`.

See also: GetVariantManager (1506)

75.12.313 SetWideStringManager

Synopsis: Set the widestring manager.

Declaration:
```
procedure SetWideStringManager(const &New: TUnicodeStringManager)
procedure SetWideStringManager(const &New: TUnicodeStringManager;
                               var Old: TUnicodeStringManager)
```

Visibility: default

Description: `SetWideStringManager` sets the current widestring manager to `New`. Optionally, it returns the currently active widestring manager in `Old`.

WideStrings are implemented in different ways on different platforms. Therefore, the Free Pascal Runtime library has no fixed implementation of widestring routines. Instead, it defines a WideString manager record, with callbacks that can be set to an implementation which is most efficient on the current platform. On windows, standard Windows routines will be used. On Unix and Linux, an implementation based on the C library is available (in unit cwstring).

It is possible to implement a custom widestring manager, optimized for the current application, without having to recompile the complete Run-Time Library.

See also: TWideStringManager (1440)

75.12.314 ShortCompareText

Synopsis: Compare 2 shortstrings.

Declaration:
```
function ShortCompareText(const S1: shortstring; const S2: shortstring)
                                 : SizeInt
```

Visibility: default

Description: `ShortCompareText` compares two shortstrings, `S1` and `S2`, and returns the following result:

<0if `S1<S2`.

0if `S1=S2`.

>0if `S1>S2`.

The comparison of the two strings is case-insensitive. The function does not take internationalization settings into account, it simply compares ASCII values.

Errors: None.

See also: CompareText (1690)

75.12.315 Sin

Synopsis: Calculate sine of angle.

Declaration:
```
function Sin(d: ValReal) : ValReal
```

Visibility: default

Description: `Sin` returns the sine of its argument `X`, where `X` is an angle in radians. If the absolute value of the argument is larger than 2$\hat{6}$3, then the result is undefined.

Errors: None.

See also: Cos (1474), Pi (1542), Exp (1490), Ln (1530)

Listing: ./examples/refex/ex62.pp

```
Program Example62;

{ Program to demonstrate the Sin function. }

begin
  WriteLn (Sin(Pi):0:1);    { Prints 0.0 }
  WriteLn (Sin(Pi/2):0:1); { Prints 1.0 }
end.
```

75.12.316 SizeOf

Synopsis: Return size (in bytes) of a variable or type.

Declaration: `function SizeOf(X: TAnyType) : LongInt`

Visibility: default

Description: `SizeOf` returns the size, in bytes, of any variable or type-identifier.

Remark This isn't really a RTL function. Its result is calculated at compile-time, and hard-coded in the executable.

Errors: None.

See also: Addr (1448), BitSizeOf (1362)

Listing: ./examples/refex/ex63.pp

```
Program Example63;

{ Program to demonstrate the SizeOf function. }
Var
  I : Longint;
  S : String [10];

begin
  WriteLn (SizeOf(I));   { Prints 4  }
  WriteLn (SizeOf(S));   { Prints 11 }
end.
```

75.12.317 SizeofResource

Synopsis: Return the size of a particular resource.

Declaration:
```
function SizeofResource(ModuleHandle: TFPResourceHMODULE;
                        ResHandle: TFPResourceHandle) : LongWord
```

Visibility: default

Description: `SizeOfResource` returns the size of the resource identified by `ResHandle` in module identified by `ModuleHandle`. `ResHandle` should be obtained from a call to LoadResource (1531)

Errors: In case of an error, 0 is returned.

See also: FindResource (1496), FreeResource (1500), LoadResource (1531), LockResource (1532), UnlockResource (1594), FreeResource (1500)

75.12.318 Slice

Synopsis: Return part of an array.

Declaration: `function Slice(const A: ArrayType; ACount: Integer) : ArrayType2`

Visibility: default

Description: `Slice` returns the first `ACount` elements from the array `A`. It returns an array with the same element type as `A`, but this array is not assignment compatible to any other array, and can therefor only be used in open array arguments to functions.

See also: Length (1528), SetLength (1566)

Listing: ./examples/refex/ex113.pp

```
Program Example113;

{ Program to demonstrate the Slice function. }

procedure ShowArray(const A: array of Integer);
var
  I: Integer;
begin
  for I := Low(A) to High(A) do
    WriteLn(I, ' : ', A[I]);
end;

begin
  ShowArray(Slice([1,2,3,4],2));
end.
```

75.12.319 Space

Synopsis: Return a string of spaces.

Declaration: `function Space(b: Byte) : shortstring`

Visibility: default

Description: `Space` returns a shortstring with length `B`, consisting of spaces.

See also: StringOfChar (1580)

75.12.320 SPtr

Synopsis: Return current stack pointer.

Declaration: `function SPtr : Pointer`

Visibility: default

Description: `Sptr` returns the current stack pointer.

Errors: None.

See also: SSeg (1577)

Listing: ./examples/refex/ex64.pp

```
program Example64;

{ Program to demonstrate the sptr function. }

var p:ptruint;

begin
  p:=ofs(stackbottom); { P Contains now the current stack position. }
end.
```

75.12.321 Sqr

Synopsis: Calculate the square of a value.

Declaration:
```
function Sqr(l: LongInt) : LongInt
function Sqr(l: Int64) : Int64
function Sqr(l: QWord) : QWord
function Sqr(d: ValReal) : ValReal
```

Visibility: default

Description: `Sqr` returns the square of its argument `X`.

Errors: None.

See also: Sqrt (1576), Ln (1530), Exp (1490)

Listing: ./examples/refex/ex65.pp

```
Program Example65;

{ Program to demonstrate the Sqr function. }
Var i : Integer;

begin
  For i:=1 to 10 do
    writeln (Sqr(i):3);
end.
```

75.12.322 Sqrt

Synopsis: Calculate the square root of a value.

Declaration:
```
function Sqrt(d: ValReal) : ValReal
```

Visibility: default

Description: `Sqrt` returns the square root of its argument `X`, which must be positive.

Errors: If `X` is negative, then a run-time error is generated.

See also: Sqr (1576), Ln (1530), Exp (1490)

Listing: ./examples/refex/ex66.pp

```
Program Example66;

{ Program to demonstrate the Sqrt function. }

begin
  WriteLn (Sqrt(4):0:3); { Prints 2.000 }
  WriteLn (Sqrt(2):0:3); { Prints 1.414 }
end.
```

75.12.323 SSeg

Synopsis: Return stack segment register value.

Declaration: `function SSeg : Word`

Visibility: default

Description: `SSeg` returns the Stack Segment. This function is only supported for compatibility reasons, as `Sptr` returns the correct contents of the stackpointer.

Errors: None.

See also: Sptr (1575)

Listing: ./examples/refex/ex67.pp

```
Program Example67;

{ Program to demonstrate the SSeg function. }
Var W : Longint;

begin
  W:=SSeg;
end.
```

75.12.324 StackTop

Synopsis: Top location of the stack.

Declaration: `function StackTop : Pointer`

Visibility: default

Description: `StackTop` contains the top of the stack for the current process. It is used to check the heap on some operating systems, and is set by the system unit initialization code. Do not use or modify this value.

See also: StackBottom (1445), StackLength (1445)

75.12.325 Str

Synopsis: Convert a numerical or enumeration value to a string.

Declaration: `procedure Str(var X: TNumericType; var S: string)`

Visibility: default

Description: `Str` returns a string which represents the value of X. X can be any numerical or enumerated type. The actual declaration of `Str` is not according to pascal syntax, and should be

```
procedure Str(const X: TNumericType[:NumPlaces[:Decimals]];var S: String)
```

Where the optional `NumPlaces` and `Decimals` specifiers control the formatting of the string: `NumPlaces` gives the total width of the string, and `Decimals` the number of decimals after the decimal separator char.

`Str` can also be used to convert an enumerated type value to a string representation of the declared enumeration value. That means that the following will work:

```
Type
  TMyEnum = (One,Two);

Var
  S : String;
begin
  Str(One,S);
  Writeln(S);
end.
```

This will write `One` on the screen, which is consistent with the following - equivalent - program:

```
Type
  TMyEnum = (One,Two);

Var
  S : String;
  E : TMyEnum;
begin
  E:=One;
  Str(E,S);
  Writeln(S);
end.
```

For scoped enumerated types, only the value is written, which means the following program will have the same output:

```
{$SCOPEDENUMS+}
Type
  TMyEnum = (One,Two);

Var
  S : String;

begin
  Str(One,S);
  Writeln(S);
end.
```

Errors: None.

See also: Val (1598)

Listing: ./examples/refex/ex68.pp

```
Program Example68;

{ Program to demonstrate the Str function. }
Var S : String;

Function IntToStr (I : Longint) : String;

Var S : String;

begin
 Str (I,S);
 IntToStr:=S;
end;

begin
  S:='*'+IntToStr(-233)+'*';
  Writeln (S);
end.
```

75.12.326 StringCodePage

Synopsis: Get the code page of a string.

Declaration:
```
function StringCodePage(const S: RawByteString) : TSystemCodePage
                        ; Overload
function StringCodePage(const S: UnicodeString) : TSystemCodePage
                        ; Overload
```

Visibility: default

Description: `StringCodePage` returns the code page of a string (`S`), regardless of the string type. It accesses the internal structures of the string to retrieve this information. For an empty string, DefaultSystemCodePage (1443) is returned.

See also: DefaultSystemCodePage (1443), StringElementSize (1579), StringRefCount (1580), SetCodePage (1565)

75.12.327 StringElementSize

Synopsis: Get the character size of a string.

Declaration:
```
function StringElementSize(const S: RawByteString) : Word;  Overload
function StringElementSize(const S: UnicodeString) : Word;  Overload
```

Visibility: default

Description: `StringCodePage` returns the character size of a string (`S`), regardless of the string type. It accesses the internal structures of the string to retrieve this information. For an empty string, `SizeOf(AnsiChar)` (normally 1) is returned.

See also: StringCodePage (1579), StringRefCount (1580)

75.12.328 StringOfChar

Synopsis: Return a string consisting of 1 character repeated N times.

Declaration:
```
function StringOfChar(c: AnsiChar; l: SizeInt) : AnsiString
function StringOfChar(c: UnicodeChar; l: SizeInt) : UnicodeString
```

Visibility: default

Description: `StringOfChar` creates a new String of length `l` and fills it with the character `c`.

It is equivalent to the following calls:

```
SetLength(StringOfChar,l);
FillChar(Pointer(StringOfChar)^,Length(StringOfChar),c);
```

Errors: None.

See also: SetLength (1566)

Listing: ./examples/refex/ex97.pp

```
Program Example97;

{$H+}

{ Program to demonstrate the StringOfChar function. }

Var S : String;

begin
  S:=StringOfChar(' ',40)+'Aligned at column 41.';
  Writeln(s);
end.
```

75.12.329 StringRefCount

Synopsis: Get the reference count of a string.

Declaration:
```
function StringRefCount(const S: RawByteString) : SizeInt;  Overload
function StringRefCount(const S: UnicodeString) : SizeInt;  Overload
```

Visibility: default

Description: `StringRefCount` returns the reference count of a string (`S`), regardless of the string type. It accesses the internal structures of the string to retrieve this information. For an empty string, 0 is returned.

See also: StringCodePage (1579), StringElementSize (1579)

75.12.330 StringToPPChar

Synopsis: Split string in list of null-terminated strings.

Declaration:
```
function StringToPPChar(var S: AnsiString; ReserveEntries: Integer)
                        : PPChar
function StringToPPChar(S: PChar; ReserveEntries: Integer) : PPChar
```

Visibility: default

Description: `StringToPPChar` splits the string `S` in words, replacing any whitespace with zero characters. It returns a pointer to an array of pchars that point to the first letters of the words in `S`. This array is terminated by a `Nil` pointer.

The function does *not* add a zero character to the end of the string unless it ends on whitespace.

The function reserves memory on the heap to store the array of `PChar`; The caller is responsible for freeing this memory.

This function is only available on certain platforms.

Errors: None.

See also: ArrayStringToPPchar (1450)

75.12.331 StringToUnicodeChar

Synopsis: Convert an ansistring to a null-terminated array of Unicode characters.

Declaration:
```
function StringToUnicodeChar(const Src: RawByteString;
                             Dest: PUnicodeChar; DestSize: SizeInt)
                              : PUnicodeChar
```

Visibility: default

Description: `StringToUnicodeChar` converts the ansistring `S` to a unicodestring and places the result in `Dest`. The size of the memory location pointed to by `Dest` must be given in `DestSize`. If the result string is longer than the available size, the result string will be truncated.

The function always returns `Dest`.

Errors: No check is performed to see whether `Dest` points to a valid memory location.

See also: UnicodeCharToString (1593), UnicodeCharLenToString (1592)

75.12.332 StringToWideChar

Synopsis: Convert a string to an array of widechars.

Declaration:
```
function StringToWideChar(const Src: RawByteString; Dest: PWideChar;
                          DestSize: SizeInt) : PWideChar
```

Visibility: default

Description: `StringToWideChar` converts a single-byte-character string `Src` to a null-terminated array of WideChars. The destination for this array is pointed to by `Dest`, and contains room for at least `DestSize` widechars.

Errors: No validity checking is performed on `Dest`.

See also: WideCharToString (1601), WideCharToStrVar (1601), WideCharLenToStrVar (1600), WideCharLenToString (1600)

75.12.333 StrLen

Synopsis: Length of a null-terminated string.

Declaration: `function StrLen(p: PChar) : SizeInt`

Visibility: default

Description: Returns the length of the null-terminated string `P`.

Errors: None.

75.12.334 StrPas

Synopsis: Convert a null-terminated string to a shortstring.

Declaration: `function StrPas(p: PChar) : shortstring`

Visibility: default

Description: Converts a null terminated string in `P` to a Pascal string, and returns this string. The string is truncated at 255 characters.

Errors: None.

75.12.335 subtract(variant,variant):variant

Synopsis: Implement subtraction `(-)` operation on variants.

Declaration: `operator -(const op1: variant; const op2: variant) : variant`

Visibility: default

Description: The implementation of the subtraction − operation is delegated to the variant manager with operation `opSubtract`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator -(variant, variant): variant (1362)

75.12.336 Succ

Synopsis: Return next element of ordinal type.

Declaration: `function Succ(X: TOrdinal) : TOrdinal`

Visibility: default

Description: `Succ` returns the element that succeeds the element that was passed to it. If it is applied to the last value of the ordinal type, and the program was compiled with range checking on (`{$R+}`), then a run-time error will be generated.

for an example, see Ord (1540).

Errors: Run-time error 201 is generated when the result is out of range.

See also: Ord (1540), Pred (1545), High (1511), Low (1534)

75.12.337 SuspendThread

Synopsis: Suspend a running thread.

Declaration:
```
function SuspendThread(threadHandle: TThreadID) : DWord
```

Visibility: default

Description: `SuspendThread` suspends a running thread. The thread is identified with it's handle or ID `threadHandle`.

The function returns zero if successful. A nonzero return value indicates failure.

Errors: If a failure occurred, a nonzero result is returned. The meaning is system dependent.

See also: ResumeThread (1553), KillThread (1527)

75.12.338 Swap

Synopsis: Swap high and low bytes/words of a variable.

Declaration:
```
function Swap(X: Word) : Word
function Swap(X: Integer) : Integer
function Swap(X: LongInt) : LongInt
function Swap(X: Cardinal) : Cardinal
function Swap(X: QWord) : QWord
function Swap(X: Int64) : Int64
```

Visibility: default

Description: `Swap` swaps the high and low order bytes of `X` if `X` is of type `Word` or `Integer`, or swaps the high and low order words of `X` if `X` is of type `Longint` or `Cardinal`. The return type is the type of `X`

Errors: None.

See also: Lo (1530), Hi (1510)

Listing: ./examples/refex/ex69.pp

```
Program Example69;

{ Program to demonstrate the Swap function. }
Var W : Word;
    L : Longint;

begin
  W:=$1234;
  W:=Swap(W);
  if W<>$3412 then
    writeln ('Error when swapping word !');
  L:=$12345678;
  L:=Swap(L);
  if L<>$56781234 then
    writeln ('Error when swapping Longint !');
end.
```

75.12.339 SwapEndian

Synopsis: Swap endianness of the argument.

Declaration:
```
function SwapEndian(const AValue: SmallInt) : SmallInt
function SwapEndian(const AValue: Word) : Word
function SwapEndian(const AValue: LongInt) : LongInt
function SwapEndian(const AValue: DWord) : DWord
function SwapEndian(const AValue: Int64) : Int64
function SwapEndian(const AValue: QWord) : QWord
```

Visibility: default

Description: `SwapEndian` will swap the endianness of the bytes in its argument.

Errors: None.

See also: hi (1510), lo (1530), swap (1583), BEToN (1458), NToBE (1538), NToLE (1538), LEToN (1529)

75.12.340 SysAllocMem

Synopsis: System memory manager: Allocate memory.

Declaration:
```
function SysAllocMem(size: PtrUInt) : Pointer
```

Visibility: default

Description: `SysFreemSize` is the system memory manager implementation for AllocMem (1449)

See also: AllocMem (1449)

75.12.341 SysAssert

Synopsis: Standard Assert failure implementation.

Declaration:
```
procedure SysAssert(const Msg: ShortString; const FName: ShortString;
                    LineNo: LongInt; ErrorAddr: Pointer)
```

Visibility: default

Description: `SysAssert` is the standard implementation of the assertion failed code. It is the default value of the `AssertErrorProc` constant. It will print the assert message `Msg` together with the filename `FName` and linenumber `LineNo` to standard error output (`StdErr`) and will halt the program with exit code 227. The error address `ErrorAddr` is ignored.

See also: AssertErrorProc (1369)

75.12.342 SysBacktraceStr

Synopsis: Format an address suitable for inclusion in a backtrace.

Declaration:
```
function SysBacktraceStr(Addr: CodePointer) : ShortString
```

Visibility: default

Description: `SysBackTraceStr` will create a string representation of the address `Addr`, suitable for inclusion in a stack backtrace.

Errors: None.

75.12.343 SysFlushStdIO

Synopsis: Flush all standard IO file descriptors.

Declaration: `procedure SysFlushStdIO`

Visibility: default

Description: `SysFlushStdIO` calls flush (1497) on all standard file descriptors: output (1445), stdout (1445), stderr (1445), erroutput (1444)

See also: output (1445), stdout (1445), stderr (1445), erroutput (1444), flush (1497)

75.12.344 SysFreemem

Synopsis: System memory manager free routine.

Declaration: `function SysFreemem(p: pointer) : PtrUInt`

Visibility: default

Description: `SysFreeem` is the system memory manager implementation for FreeMem (1499)

See also: FreeMem (1499)

75.12.345 SysFreememSize

Synopsis: System memory manager free routine.

Declaration: `function SysFreememSize(p: pointer; Size: PtrUInt) : PtrUInt`

Visibility: default

Description: `SysFreemSize` is the system memory manager implementation for FreeMem (1499)

See also: MemSize (1535)

75.12.346 SysGetFPCHeapStatus

Synopsis: Return the status of the FPC heapmanager.

Declaration: `function SysGetFPCHeapStatus : TFPCHeapStatus`

Visibility: default

Description: `SysGetFPCHeapStatus` returns the status of the default FPC heapmanager. It is set as the default value of the corresponding GetFPCHeapStatus (1502) function.

Errors: None. The result of this function is bogus information if the current heapmanager is not the standard FPC heapmanager.

See also: GetFPCHeapStatus (1502)

75.12.347 SysGetHeapStatus

Synopsis: System implementation of GetHeapStatus (1502).

Declaration: `function SysGetHeapStatus : THeapStatus`

Visibility: default

Description: `SysGetHeapStatus` is the system implementation of the GetHeapStatus (1502) call.

See also: GetHeapStatus (1502)

75.12.348 SysGetmem

Synopsis: System memory manager memory allocator.

Declaration: `function SysGetmem(Size: PtrUInt) : Pointer`

Visibility: default

Description: `SysGetmem` is the system memory manager implementation for GetMem (1502)

See also: GetMem (1502), GetMemory (1503)

75.12.349 SysInitExceptions

Synopsis: Initialize exceptions.

Declaration: `procedure SysInitExceptions`

Visibility: default

Description: `SysInitExceptions` initializes the exception system. This procedure should never be called directly, it is taken care of by the RTL.

75.12.350 SysInitFPU

Synopsis: Initialize the FPU.

Declaration: `procedure SysInitFPU`

Visibility: default

Description: `SysInitFPU` initializes (resets) the floating point unit, if one is available. It is called for instance when a new thread is started.

See also: BeginThread (1458)

75.12.351 SysInitStdIO

Synopsis: Initialize standard input and output.

Declaration: `procedure SysInitStdIO`

Visibility: default

Description: `SysInitStdIO` initializes the standard input and output files: Output (1445), Input (1444) and StdErr (1445). This routine is called by the initialization code of the system unit, there should be no need to call it directly.

75.12.352 SysMemSize

Synopsis: System memory manager: free size.

Declaration: `function SysMemSize(p: pointer) : PtrUInt`

Visibility: default

Description: `SysFreemSize` is the system memory manager implementation for MemSize (1535)

See also: MemSize (1535)

75.12.353 SysReAllocMem

Synopsis: System memory manager: Reallocate memory.

Declaration: `function SysReAllocMem(var p: pointer; size: PtrUInt) : Pointer`

Visibility: default

Description: `SysReallocMem` is a help routine for the system memory manager implementation for ReAllocMem (1551).

See also: ReAllocMem (1551)

75.12.354 SysResetFPU

Synopsis: Reset the floating point unit.

Declaration: `procedure SysResetFPU`

Visibility: default

Description: `SysResetFPU` resets the floating point unit. There should normally be no need to call this unit; the compiler itself takes care of this.

75.12.355 SysSetCtrlBreakHandler

Synopsis: System CTRL-C handler.

Declaration:
```
function SysSetCtrlBreakHandler(Handler: TCtrlBreakHandler)
                                    : TCtrlBreakHandler
```

Visibility: default

Description: `SysSetCtrlBreakHandler` sets the CTRL-C handler to the `Handler` callback, and returns the previous value of the handler.

See also: TCtrlBreakHandler (1413)

75.12.356 SysTryResizeMem

Synopsis: System memory manager: attempt to resize memory.

Declaration: `function SysTryResizeMem(var p: pointer; size: PtrUInt) : Boolean`

Visibility: default

Description: `SysTryResizeMem` is a help routine for the system memory manager implementation for ReAllocMem (1551), SysReAllocMem (1587)

See also: SysReAllocMem (1587), ReAllocMem (1551)

75.12.357 ThreadGetPriority

Synopsis: Return the priority of a thread.

Declaration:
```
function ThreadGetPriority(threadHandle: TThreadID) : LongInt
```

Visibility: default

Description: `ThreadGetPriority` returns the priority of thread `TThreadID` to `Prio`. The returned priority is a value between -15 and 15.

Errors: None.

See also: ThreadSetPriority (1588)

75.12.358 ThreadSetPriority

Synopsis: Set the priority of a thread.

Declaration:
```
function ThreadSetPriority(threadHandle: TThreadID; Prio: LongInt)
                                  : Boolean
```

Visibility: default

Description: `ThreadSetPriority` sets the priority of thread `TThreadID` to `Prio`. Priority is a value between -15 and 15.

Errors: None.

See also: ThreadGetPriority (1588)

75.12.359 ThreadSwitch

Synopsis: Signal possibility of thread switch.

Declaration:
```
procedure ThreadSwitch
```

Visibility: default

Description: `ThreadSwitch` signals the operating system that the thread should be suspended and that another thread should be executed.

This call is a hint only, and may be ignored.

See also: SuspendThread (1583), ResumeThread (1553), KillThread (1527)

75.12.360 ToSingleByteFileSystemEncodedFileName

Synopsis: Convert string to encoding for use in single-byte file system API.

Declaration:
```
function ToSingleByteFileSystemEncodedFileName(const Str: UnicodeString)
                                                : RawByteString
function ToSingleByteFileSystemEncodedFileName
                                               (const arr: Array of WideChar)
                                                : RawByteString
function ToSingleByteFileSystemEncodedFileName(const Str: RawByteString)
                                                : RawByteString
```

Visibility: default

Description: `ToSingleByteFileSystemEncodedFileName` converts the argument (`Str` or `Arr`) to a single-byte string, encoded using the codepage used by the single-byte file system API.

This routine is simply an auxiliary routine, which converts the argument to a single-byte string using DefaultFileSystemCodePage (1442) as a codepage.

See also: DefaultFileSystemCodePage (1442)

75.12.361 Trunc

Synopsis: Truncate a floating point value.

Declaration: `function Trunc(d: ValReal) : Int64`

Visibility: default

Description: `Trunc` returns the integer part of `X`, which is always smaller than (or equal to) `X` in absolute value.

Errors: An invalid floating point operation error may occur in one of several conditions:

- `Infinity` or `NegInfinity` is specified.
- `Nan` is specified.
- `X` is out of the valid range of integers.

See also: Frac (1499), Int (1521), Round (1557)

Listing: ./examples/refex/ex70.pp

```
Program Example70;

{ Program to demonstrate the Trunc function. }

begin
  WriteLn (Trunc(123.456)); { Prints 123  }
  WriteLn (Trunc(-123.456)); { Prints -123 }
  WriteLn (Trunc(12.3456)); { Prints 12   }
  WriteLn (Trunc(-12.3456)); { Prints -12  }
end.
```

Listing: ./examples/refex/ex124.pp

```
program ex124;

{
  Example to show various error conditions for the Trunc() function.
  SysUtils is used to convert the runtime error to an exception that can be caught
  Math is used to have access to Nan and (Neg)Infinity

}

{$mode objfpc}
{$h+}
uses

  sysutils, math;
var
  D: Double;
  Q: QWord;
```

```
  L: Int64;
begin
  D := 2.0 * High(QWord);
  try
    Q := Trunc(D);
  except
    on E: Exception do writeln(E.ClassName,': ',E.Message);   //EInvalidOp: Invalid floating p
  end;
  D := NaN;
  try
    Q := Trunc(D);
  except
    on E: Exception do writeln(E.ClassName,': ',E.Message);   //EInvalidOp: Invalid floating p
  end;
  D := Infinity;
  try
    Q := Trunc(D);
  except
    on E: Exception do writeln(E.ClassName,': ',E.Message);   //EInvalidOp: Invalid floating p
  end;
  D := NegInfinity;
  try
    Q := Trunc(D);
  except
    on E: Exception do writeln(E.ClassName,': ',E.Message);   //EInvalidOp: Invalid floating p
  end;
  D := -2.0 * High(QWord);
  try
    L := Trunc(D);
  except
    on E: Exception do writeln(E.ClassName,': ',E.Message);   //EInvalidOp: Invalid floating p
  end;
end.
```

75.12.362 Truncate

Synopsis: Truncate the file at position.

Declaration: `procedure Truncate(var F: File)`

Visibility: default

Description: `Truncate` truncates the (opened) file `F` at the current file position.

Errors: Depending on the state of the `{$I}` switch, a runtime error can be generated if there is an error. In the `{$I-}` state, use `IOResult` to check for errors.

See also: Append (1449), Filepos (1491), Seek (1562)

Listing: ./examples/refex/ex71.pp

```
Program Example71;

{ Program to demonstrate the Truncate function. }

Var F : File of longint;
    I,L : Longint;
```

```
begin
  Assign (F,'test.tmp');
  Rewrite (F);
  For I:=1 to 10 Do
    Write (F,I);
  Writeln ('Filesize before Truncate : ',FileSize(F));
  Close (f);
  Reset (F);
  Repeat
    Read (F,I);
  Until i=5;
  Truncate (F);
  Writeln ('Filesize after Truncate  : ',Filesize(F));
  Close (f);
end.
```

75.12.363 TryEnterCriticalSection

Synopsis: Try entering a critical section.

Declaration: `function TryEnterCriticalSection(var cs: TRTLCRITICALSECTION) : LongInt`

Visibility: default

Description: `TryEnterCriticalsection` attempts to enter critical section `cs`. It returns at once. The return value is zero if another thread owns the critical section, or nonzero if the current thread already owns or successfully obtained the critical section.

75.12.364 TypeInfo

Synopsis: Return pointer to type information for type.

Declaration: `function TypeInfo(const T: AnyType) : Pointer`

Visibility: default

Description: `TypeInfo` is a compiler intrinsic: it returns a pointer to the generated type information (RTTI) for the type `T`. If no type information was yet generated for the type, this statement will ensure that type information is available, i.e. the result is always non-nil.

See also: Default (1476), TypeOf (1591), GetTypeKind (1505), Initialize (1517), Finalize (1495)

75.12.365 TypeOf

Synopsis: Return pointer to VMT of an object.

Declaration: `function TypeOf(T: TObjectType) : Pointer`

Visibility: default

Description: `TypeOf` is a compiler intrinsic: it returns a pointer to the VMT of the object type `T`.

See also: Default (1476), TypeInfo (1591), GetTypeKind (1505)

75.12.366 UCS4StringToUnicodeString

Synopsis: Convert a UCS-4 encoded string to a Unicode string.

Declaration:
```
function UCS4StringToUnicodeString(const s: UCS4String) : UnicodeString
```

Visibility: default

Description: `UCS4StringToUnicodeString` converts the UCS-4 encoded string `S` to a Unicode string and returns the resulting string.

This function requires the widestring manager.

See also: UnicodeStringToUCS4String (1593)

75.12.367 UCS4StringToWideString

Synopsis:

Declaration:
```
function UCS4StringToWideString(const s: UCS4String) : WideString
```

Visibility: default

Description:

75.12.368 Unassigned

Synopsis: Unassigned variant.

Declaration:
```
function Unassigned : Variant
```

Visibility: default

75.12.369 UnicodeCharLenToString

Synopsis: Convert a memory buffer with Unicode characters to a unicodestring.

Declaration:
```
function UnicodeCharLenToString(S: PUnicodeChar; Len: SizeInt)
                                    : UnicodeString
```

Visibility: default

Description: `UnicodeCharLenToString` converts the Unicode characters in buffer `S` with at most `len` bytes length, to a unicodestring and returns the result.

This function requires the use of a widestring manager.

Errors: No checking is done to see if the pointer `S` or length `len` are valid.

See also: StringToUnicodeChar (1581), UnicodeCharToString (1593)

75.12.370 UnicodeCharLenToStrVar

Synopsis: Convert a memory buffer with Unicode characters to an ansistring.

Declaration:
```
procedure UnicodeCharLenToStrVar(Src: PUnicodeChar; Len: SizeInt;
                                 out Dest: UnicodeString)
procedure UnicodeCharLenToStrVar(Src: PUnicodeChar; Len: SizeInt;
                                 out Dest: AnsiString)
```

Visibility: default

Description: `UnicodeCharLenToString` converts the Unicode characters in buffer `S` with at most `len` bytes length, to an ansistring and returns the result in `Dest`

This function does the same as UnicodeCharLenToString (1592).

Errors: No checking is done to see if the pointer `S` or length `len` are valid.

See also: StringToUnicodeChar (1581), UnicodeCharToString (1593), UnicodeCharLenToString (1592), UnicodeCharToStrVar (1593)

75.12.371 UnicodeCharToString

Synopsis: Convert Unicode character to string.

Declaration: `function UnicodeCharToString(S: PUnicodeChar) : UnicodeString`

Visibility: default

Description: `UnicodeCharToString` converts a null-word-terminated array of Unicode characters in `S` to a Unicode string value. It simply calls UnicodeCharLenToString (1592) with the length of the string `S`.

This function requires the use of a widestring manager.

Errors: No checking is done to see if the pointer `S` is valid.

See also: StringToUnicodeChar (1581), UnicodeCharLenToString (1592), WidestringManager (1446)

75.12.372 UnicodeCharToStrVar

Synopsis: Convert a null-terminated memory buffer with Unicode characters to an ansistring.

Declaration: `procedure UnicodeCharToStrVar(S: PUnicodeChar; out Dest: AnsiString)`

Visibility: default

Description: `UnicodeCharLenToString` converts the Unicode characters in buffer `S` up to the first null word, to an ansistring and returns the result in `Dest`

This function does the same as UnicodeCharToString (1593).

Errors: No checking is done to see if the pointer `S` is valid.

See also: StringToUnicodeChar (1581), UnicodeCharToString (1593), UnicodeCharLenToString (1592), UnicodeCharToString (1593)

75.12.373 UnicodeStringToUCS4String

Synopsis: Convert a Unicode string to a UCS-4 string.

Declaration: `function UnicodeStringToUCS4String(const s: UnicodeString) : UCS4String`

Visibility: default

Description: `UnicodeStringToUCS4String` converts a Unicode string `S` to a UCS-4 encoded string, and returns the resulting string.

This function requires the widestring manager.

See also: UCS4StringToUnicodeString (1592)

75.12.374 UnicodeToUtf8

Synopsis:

Declaration:
```
function UnicodeToUtf8(Dest: PChar; Source: PUnicodeChar;
                       MaxBytes: SizeInt) : SizeInt
function UnicodeToUtf8(Dest: PChar; MaxDestBytes: SizeUInt;
                       Source: PUnicodeChar; SourceChars: SizeUInt)
                        : SizeUInt
```

Visibility: default

Description:

75.12.375 UniqueString

Synopsis: Make sure reference count of string is 1.

Declaration:
```
procedure UniqueString(var S: RawByteString)
procedure UniqueString(var S: UnicodeString)
procedure UniqueString(var S: WideString)
```

Visibility: default

Description: `UniqueString` ensures that the ansistring `S` has reference count 1. It makes a copy of `S` if this is necessary, and returns the copy in `S`

Errors: None.

75.12.376 UnloadLibrary

Synopsis: Unload a previously loaded library.

Declaration:
```
function UnloadLibrary(Lib: TLibHandle) : Boolean
```

Visibility: default

Description: `UnloadLibrary` unloads a previously loaded library (specified by the handle `lib`). The call returns `True` if successful, `False` otherwise.

Errors: On error, `False` is returned.

See also: LoadLibrary (1531), GetProcedureAddress (1504)

75.12.377 UnlockResource

Synopsis: Unlock a previously locked resource.

Declaration:
```
function UnlockResource(ResData: TFPResourceHGLOBAL) : LongBool
```

Visibility: default

Description: `UnlockResource` unlocks a previously locked resource. Note that this function does not exist on windows, it's only needed on other platforms.

Errors: The function returns `False` if it failed.

See also: FindResource (1496), FreeResource (1500), SizeofResource (1574), LoadResource (1531), lockResource (1532), FreeResource (1500)

75.12.378 UnPack

Synopsis: Create unpacked array from packed array.

Declaration:
```
procedure UnPack(const Z: PackedArrayType; out A: UnpackedArrayType;
                 StartIndex: TIndexType)
```

Visibility: default

Description: `UnPack` will copy the elements of a packed array (`Z`) to an unpacked array (`A`). All elements in `Z` are copied to `A`, starting at index `StartIndex` in `A`. The type of the index variable `StartIndex` must match the type of the index of `A`.

Obviously, the type of the elements of the arrays `A` and `Z` must match.

See also: Pack (1541)

75.12.379 UpCase

Synopsis: Convert a string to all uppercase.

Declaration:
```
function UpCase(const s: shortstring) : shortstring
function UpCase(c: Char) : Char
function UpCase(const s: ansistring) : ansistring
function UpCase(const s: UnicodeString) : UnicodeString
function UpCase(c: UnicodeChar) : UnicodeChar
function UpCase(const s: WideString) : WideString
```

Visibility: default

Description: `Upcase` returns the uppercase version of its argument `C`. If its argument is a string, then the complete string is converted to uppercase. The type of the returned value is the same as the type of the argument.

`UpCase` does not change the number of characters in the string.

Errors: None.

See also: Lowercase (1534)

Listing: ./examples/refex/ex72.pp

```
program Example72;

{ Program to demonstrate the upcase function. }

var c:char;

begin
  for c:='a' to 'z' do
    write(upcase(c));
  Writeln;
  { This doesn't work in TP, but it does in Free Pascal }
  Writeln(upcase('abcdefghijklmnopqrstuvwxyz'));
end.
```

75.12.380 Utf8CodePointLen

Synopsis: Length of an UTF-8 codepoint.

Declaration:
```
function Utf8CodePointLen(P: PAnsiChar; MaxLookAhead: SizeInt;
                          IncludeCombiningDiacriticalMarks: Boolean)
                           : SizeInt
```

Visibility: default

Description: `Utf8CodePointLen` returns the length of the UTF-8 codepoint starting at the beginning of `P`. It will look at at most `MaxLookAhead` bytes to do create this codepoint. If `IncludeCombiningDiacriticalMarks` is true, combining diacritical marks trailing the first codepoint (which itself can also be such a mark) will be considered to be part of the codepoint.

If the function returns a value > 0, then this is the number of bytes occupied by the codepoint and, if requested, the trailing combining diacritical marks. If the result = 0, this means that all bytes within the requested `MaxLookAhead` could be part of a single valid codepoint and, if requested, its trailing diacritical marks, but that the codepoint is incomplete and more bytes need to be looked at. If the result is < 0, then the function determined that the codepoint was invalid after processing the number of bytes equal to the absolute value of the function result.

If `IncludeCombiningDiacriticalMarks` is `True`, then

- If the function processes all `MaxLookAhead` bytes, it will return the value `MaxLookAhead` rather than 0, even though in theory more combining diacritical marks might follow if more bytes would be looked at. Therefore, in order to ascertain that all combining diacritical marks are processed, pass all bytes at once to this function.
- If an invalid sequence is detected while processing a potential combining diacritical mark after a valid codepoint has been found already, the function will return the length of this valid codepoint (plus that of any preceding valid combining diacritical marks) as a positive value. The idea is that this invalid sequence at the end is by definition not a combining diacritical mark (since all of those are valid sequences) and hence should not render the preceding codepoint invalid.

Errors: None.

75.12.381 UTF8Decode

Synopsis: Convert an UTF-8 encoded ansistring to a unicodestring.

Declaration:
```
function UTF8Decode(const s: RawByteString) : UnicodeString
```

Visibility: default

Description: `UTF8Decode` converts the UTF-8 encoded ansistring `S` to a unicodestring and returns the resulting string. It calls the low-level Utf8ToUnicode (1597) function to do the actual work.

See also: UTF8Encode (1596), Utf8ToAnsi (1597), SetWideStringManager (1573), Utf8ToUnicode (1597)

75.12.382 UTF8Encode

Synopsis: Convert a widestring or unicodestring to an UTF-8 encoded ansistring.

Declaration:
```
function UTF8Encode(const s: RawByteString) : RawByteString
function UTF8Encode(const s: UnicodeString) : RawByteString
function UTF8Encode(const s: WideString) : RawByteString
```

Visibility: default

Description: `UTF8Encode` converts an ansistring or widestring `S` to the equivalent UTF-8 encoded Unicode string and returns this resulting string. It calls the low-level UnicodeToUTF8 (1594) function to do the actual work.

The resulting string has code page `CP_UTF8`.

See also: UTF8Decode (1596), Utf8ToAnsi (1597), UnicodeToUtf8 (1594), SetWideStringManager (1573)

75.12.383 Utf8ToAnsi

Synopsis: Convert a UTF-8 encoded Unicode string to an ansistring.

Declaration:
```
function Utf8ToAnsi(const s: RawByteString) : RawByteString
```

Visibility: default

Description: `Utf8ToAnsi` converts an utf8-encode Unicode string to an ansistring. It converts the string to a widestring and then converts the widestring to an ansistring.

For this function to work, a widestring manager must be installed.

See also: UTF8Encode (1596), UTF8Decode (1596), SetWideStringManager (1573)

75.12.384 UTF8ToString

Synopsis: Convert UTF8 to Unicode String.

Declaration:
```
function UTF8ToString(const s: RawByteString) : UnicodeString
function UTF8ToString(const S: ShortString) : unicodestring
function UTF8ToString(const S: PAnsiChar) : unicodestring
function UTF8ToString(const S: Array of AnsiChar) : unicodestring
function UTF8ToString(const S: Array of Byte) : unicodestring
```

Visibility: default

Description: `UTF8ToString` converts a UTF8 single byte string to a UTF16 encoded unicodestring. The source string `S` is interpreted as an array of UTF8 characters.

75.12.385 Utf8ToUnicode

Synopsis: Convert a buffer with UTF-8 characters to widestring characters.

Declaration:
```
function Utf8ToUnicode(Dest: PUnicodeChar; Source: PChar;
                       MaxChars: SizeInt) : SizeInt
function Utf8ToUnicode(Dest: PUnicodeChar; MaxDestChars: SizeUInt;
                       Source: PChar; SourceBytes: SizeUInt) : SizeUInt
```

Visibility: default

Description: `Utf8ToUnicode` converts the buffer in `Source` with a length of `SourceBytes` or for a maximum length of `MaxChars` (or `MaxDestChars`) widestring characters to the buffer pointed to by `Dest`.

The function returns the number of copied widestring characters.

Errors: On error, -1 is returned.

See also: UTF8Encode (1596), UTF8Decode (1596), Utf8ToAnsi (1597), SetWideStringManager (1573)

75.12.386 Val

Synopsis: Calculate numerical/enumerated value of a string.

Declaration: `procedure Val(const S: string; var V; var Code: Word)`

Visibility: default

Description: `Val` converts the value represented in the string `S` to a numerical value or an enumerated value, and stores this value in the variable `V`, which can be of any integer type (signed or unsigned, sizes from 1 to 8 bytes), floating point type (of any supported size) or any enumerated type. If the conversion is not successful, then the parameter `Code` contains the index of the character in `S` which prevented the conversion. The string `S` is allowed to contain spaces in the beginning.

The string `S` can contain a number in decimal, hexadecimal, binary or octal format, as described in the language reference. For enumerated values, the string must be the name of the enumerated value. The name is searched case insensitively.

The string can use various radices in addition to a decimal (radix 10) notation:

- `$`, `0x` or `x` can be used for hexadecimal notation. Allowed characters are `0..9` and `A..F` (case insensitive)
- `&` for octal notation. Allowed characters are `0..7`.
- `%` for binary notation. Allowed characters are `0..1`.

The conversion to enumerated exists only as of version 2.3.1 (or later) of the compiler.

Errors: If the conversion doesn't succeed, the value of `Code` indicates the position where the conversion went wrong. The value of `V` is then undefined.

See also: Str (1577)

Listing: ./examples/refex/ex74.pp

```
Program Example74;

{ Program to demonstrate the Val function. }
Var I, Code : Integer;

begin
  Val (ParamStr (1),I,Code);
  If Code<>0 then
    WriteIn ('Error at position ',code,' : ',Paramstr(1)[Code])
  else
    WriteIn ('Value : ',I);
end.
```

75.12.387 VarArrayGet

Synopsis: Get a value from a single cell of a variant array.

Declaration: `function VarArrayGet(const A: Variant; const Indices: Array of LongInt)
: Variant`

Visibility: default

Description: `VarArrayGet` returns the value in the variant array `A` at the location indicated by `Indices`. Thus the statement

```
B:=VarArrayGet(A,[2,1]);
```

is equivalent to

```
B:=A[2,1];
```

The difference is that the previous is usable when the amount of indices is not known at compile time.

Errors: If the number of indices is wrong (or out of range) an exception may be raised.

See also: VarArrayPut (1599)

75.12.388 VarArrayPut

Synopsis: Put a value in a single cell of a variant array.

Declaration:
```
procedure VarArrayPut(var A: Variant; const Value: Variant;
                      const Indices: Array of LongInt)
```

Visibility: default

Description: `VarArrayPut` puts `Value` in the variant array `A` at the location indicated by `Indices`. Thus the statement

```
VarArrayPut(A,B,[2,1]);
```

is equivalent to

```
A[2,1]:=B;
```

The difference is that the previous is usable when the amount of indices is not known at compile time.

Errors: If the number of indices is wrong (or out of range) an exception may be raised.

See also: VarArrayGet (1598)

75.12.389 VarArrayRedim

Synopsis: Redimension a variant array.

Declaration:
```
procedure VarArrayRedim(var A: Variant; HighBound: SizeInt)
```

Visibility: default

Description: `VarArrayRedim` re-sizes the first dimension of the variant array `A`, giving it a new high bound `HighBound`. Obviously, `A` must be a variant array for this function to work.

75.12.390 VarCast

Synopsis: Cast a variant to a certain type.

Declaration:
```
procedure VarCast(var dest: variant; const source: variant;
                  vartype: LongInt)
```

Visibility: default

Description: `VarCast` converts the variant in `Source` to the type indicated in `VarType` and returns the result in `dest`. The `VarType` must be one of the predefined `VarNNN` constants.

Errors: If the conversion is not possible because the value cannot be correctly casted, then a run-time error or an exception may occur.

75.12.391 WaitForThreadTerminate

Synopsis: Wait for a thread to terminate.

Declaration:
```
function WaitForThreadTerminate(threadHandle: TThreadID;
                                TimeoutMs: LongInt) : DWord
```

Visibility: default

Description: `WaitForThreadTerminate` waits for a thread to finish its execution. The thread is identified by it's handle or ID `threadHandle`. If the thread does not exit within `TimeoutMs` milliseconds, the function will return with an error value.

The function returns the exit code of the thread.

Not all platforms support the timeout parameter: the Unix platforms (with threads support based on pthreads) do not support timeout, and will wait indefinitely for the thread to exit.

See also: EndThread (1483), KillThread (1527)

75.12.392 WideCharLenToString

Synopsis: Convert a length-limited array of widechar to an unicodestring.

Declaration:
```
function WideCharLenToString(S: PWideChar; Len: SizeInt) : UnicodeString
```

Visibility: default

Description: `WideCharLenToString` converts at most `Len` widecharacters from the null-terminated widechar array `S` to an unicodestring, and returns the unicodestring.

Errors: No validity checking is performed on `S`. Passing an invalid pointer may lead to access violations.

See also: StringToWideChar (1581), WideCharToString (1601), WideCharToStrVar (1601), WideCharLenToStrVar (1600)

75.12.393 WideCharLenToStrVar

Synopsis: Convert a length-limited array of widechar to an ansistring.

Declaration:
```
procedure WideCharLenToStrVar(Src: PWideChar; Len: SizeInt;
                              out Dest: UnicodeString)
procedure WideCharLenToStrVar(Src: PWideChar; Len: SizeInt;
                              out Dest: AnsiString)
```

Visibility: default

Description: `WideCharLenToString` converts at most `Len` widecharacters from the null-terminated widechar array `Src` to an ansistring or Unicode string, and returns the resulting in `Dest`.

Errors: No validity checking is performed on `Src`. Passing an invalid pointer may lead to access violations.

See also: StringToWideChar (1581), WideCharToString (1601), WideCharToStrVar (1601), WideCharLenToString (1600)

75.12.394 WideCharToString

Synopsis: Convert a null-terminated array of widechar to an unicodestring.

Declaration:
```
function WideCharToString(S: PWideChar) : UnicodeString
```

Visibility: default

Description: `WideCharToString` converts the null-terminated widechar array `S` to an unicodestring, and returns the unicodestring.

Errors: No validity checking is performed on `Src`. Passing an invalid pointer, or an improperly terminated array may lead to access violations.

See also: StringToWideChar (1581), WideCharToStrVar (1601), WideCharLenToStrVar (1600), WideCharLenToString (1600)

75.12.395 WideCharToStrVar

Synopsis: Convert a null-terminated array of widechar to an ansistring.

Declaration:
```
procedure WideCharToStrVar(S: PWideChar; out Dest: UnicodeString)
procedure WideCharToStrVar(S: PWideChar; out Dest: AnsiString)
```

Visibility: default

Description: `WideCharToString` converts the null-terminated widechar array `S` to an ansistring or Unicode string, and returns the resulting string in `Dest`.

Errors: No validity checking is performed on `S`. Passing an invalid pointer, or an improperly terminated array may lead to access violations.

See also: StringToWideChar (1581), WideCharToString (1601), WideCharLenToStrVar (1600), WideCharLenToString (1600)

75.12.396 WideStringToUCS4String

Synopsis: Convert a widestring to a UCS-4 encoded string.

Declaration:
```
function WideStringToUCS4String(const s: WideString) : UCS4String
```

Visibility: default

Description: Convert a widestring to a UCS-4 encoded string.

75.12.397 Write

Synopsis: Write variable to a text file or standard output.

Declaration:
```
procedure Write(V1: Type1)
procedure Write(V1: Type1; V2: type2)
procedure Write(V1: Type1; V2: Type2; V3: Type3)
procedure Write(var F: Text; V1: Type1)
procedure Write(var F: Text; V1: Type1; V2: type2)
procedure Write(var F: Text; V1: Type1; V2: Type2; V3: Type3)
```

Visibility: default

Description: `Write` writes the contents of the variables `V1`, `V2`, `V3` etc. to the file `F`. `F` can be a typed file, or a `Text` file. If `F` is a typed file, then the variables `V1`, `V2` etc. must be of the same type as the type in the declaration of `F`. Untyped files are not allowed.

The `Write` command accepts an arbitrary number of arguments. The `V1`, `V2`, `V3` in the declaration here are in fact just samples, the actual number may be much higher. The types of arguments (`Type1` etc.) are limited to the following types:

- Any character type.
- Any string type (including `pchar`).
- Any ordinal type (integer, enumerated).
- The Int64 and QWord type.
- Any floating-point type (such as double, single, extended).

If the parameter `F` is omitted, standard output is assumed. If `F` is of type `Text`, then the necessary conversions are done such that the output of the variables is in human-readable format. This conversion is done for all numerical types. Strings are printed exactly as they are in memory, as well as `PChar` types.

The format of the numerical conversions can be influenced through the following modifiers: `OutputVariable: NumChars [: Decimals ]` This will print the value of `OutputVariable` with a minimum of `NumChars` characters, from which `Decimals` are reserved for the decimals. If the number cannot be represented with `NumChars` characters, `NumChars` will be increased, until the representation fits. If the representation requires less than `NumChars` characters then the output is filled up with spaces, to the left of the generated string, thus resulting in a right-aligned representation. If no formatting is specified, then the number is written using its natural length, with nothing in front of it if it's positive, and a minus sign if it's negative. Real numbers are, by default, written in scientific notation.

Remark When writing string variables, no codepage conversions are done. The string is copied as-is to the file descriptor. In particular, for console output, it is the programmer's responsibility to make sure that the codepage of the string matches the codepage of the console.

Remark Note that on MS Windows GUI applications do not have a standard output by default: Standard file descriptors are available only when the

```
{$APPPTYPE CONSOLE}
```

directive is present in the program file.

Errors: If an error occurs, a run-time error is generated. This behavior can be controlled with the `{$I}` switch.

See also: WriteLn (1603), Read (1547), Readln (1549), Blockwrite (1460)

75.12.398 WriteBarrier

Synopsis: Memory write barrier.

Declaration: `procedure WriteBarrier`

Visibility: default

Description: `WriteBarrier` is a low-level instruction to force a write barrier in the CPU: write (store) operations before and after the barrier are separate.

See also: ReadBarrier (1548), ReadDependencyBarrier (1548), ReadWriteBarrier (1550)

75.12.399 WriteLn

Synopsis: Write variable to a text file or standard output and append newline.

Declaration:
```
procedure WriteLn(V1: Type1)
procedure WriteLn(V1: Type1; V2: type2)
procedure WriteLn(V1: Type1; V2: Type2; V3: Type3)
procedure WriteLn(var F: Text; V1: Type1)
procedure WriteLn(var F: Text; V1: Type1; V2: type2)
procedure WriteLn(var F: Text; V1: Type1; V2: Type2; V3: Type3)
```

Visibility: default

Description: `WriteLn` does the same as Write (1602) for text files, and emits a Carriage Return - LineFeed character pair after that. If the parameter `F` is omitted, standard output is assumed. If no variables are specified, a newline character sequence is emitted, resulting in a new line in the file `F`.

Remark The newline character is determined by the slinebreak (1388) constant.

Remark When writing string variables, no codepage conversions are done. The string is copied as-is to the file descriptor. In particular, for console output, it is the programmer's responsibility to make sure that the codepage of the string matches the codepage of the console.

More details can be found in the Write (1602) description.

Errors: If an error occurs, a run-time error is generated. This behavior can be controlled with the `{$I}` switch.

See also: Write (1602), Read (1547), Readln (1549), Blockwrite (1460), slinebreak (1388)

Listing: ./examples/refex/ex75.pp

```
Program Example75;

{ Program to demonstrate the Write(ln) function. }

Var
  F : File of Longint;
  L : Longint;

begin
  Write ('This is on the first line ! '); { No CR/LF pair! }
  Writeln ('And this too...');
  Writeln ('But this is already on the second line...');
  Assign (f,'test.tmp');
  Rewrite (f);
  For L:=1 to 10 do
```

```
    write (F,L); { No writeln allowed here ! }
  Close (f);
end.
```

75.12.400 WriteStr

Synopsis: Write variables to a string.

Declaration: `procedure WriteStr(out S: string; Args: Arguments)`

Visibility: default

Description: `WriteStr` behaves like Write (1602), except that it stores its output in the string variable `S` instead of a file. Semantically, the `WriteStr` call is equivalent to writing the arguments to a file using the `Write` call, and then reading them into `S` using the `Read` call from the same file:

```
var
  F : Text;
begin
  Rewrite(F);
  Write(F,Args);
  Close(F);
  Reset(F);
  Read(F,S);
  Close(F);
end;
```

Obviously, the `WriteStr` call does not use a temporary file.

`WriteStr` is defined in the ISO Extended Pascal standard. More information on the allowed arguments and the possible formatting can be found in the description of Write (1602).

See also: Write (1602), ReadStr (1549), Read (1547)

75.13 TDoubleRec

```
TDoubleRec = packed record
private
  Bias = $3FF;
  function GetExp
  : QWord;
  procedure SetExp(e: QWord);
  function GetSign : Boolean
  ;
  procedure SetSign(s: Boolean);
  function GetFrac : QWord;
  procedure
  SetFrac(e: QWord);
public
  function Mantissa(IncludeHiddenBit: Boolean
  ) : QWord;
  function Fraction : ValReal;
  function Exponent : LongInt
```

```
  ;
  property Sign : Boolean;
  property Exp : QWord;
  property Frac
  : QWord;
  function SpecialType : TFloatSpecial;
  procedure BuildUp
  (const _Sign: Boolean; const _Mantissa: QWord;
  const _Exponent: LongInt);
case Byte of
0: (
public
  Bytes : Array
  [0..7] of Byte;
);
1: (
public
  Words : Array[0..3] of Word;
);
2
  : (
public
  Data : QWord;
);
3: (
public
  Value : Double;
);
end
```

`TDoubleRec` models the memory layout of a double value when using software floating point math.

75.13.1 Method overview

Page	Method	Description
1606	BuildUp	Build a double value.
1606	Exponent	Exponent of the floating point value.
1606	Fraction	Fraction of the floating point value.
1605	Mantissa	Mantissa of the floating point value.
1606	SpecialType	Is the floating point value special ?

75.13.2 Property overview

Page	Properties	Access	Description
1607	Exp	rw	Exponent bitpattern representation.
1607	Frac	rw	Fractional part of double.
1607	Sign	rw	Sign of the floating point value.

75.13.3 TDoubleRec.Mantissa

Synopsis: Mantissa of the floating point value.

Declaration: `function Mantissa(IncludeHiddenBit: Boolean) : QWord`

Visibility: public

Description: `Mantissa` returns the Mantissa part (significand bitpattern without hidden bit) of the floating point value.

See also: TDoubleRec.Fraction (1606), TDoubleRec.Exponent (1606), TDoubleRec.SpecialType (1606)

75.13.4 TDoubleRec.Fraction

Synopsis: Fraction of the floating point value.

Declaration: `function Fraction : ValReal`

Visibility: public

Description: `Fraction` returns the fraction (value after decimal) of the floating point value.

See also: TDoubleRec.Mantissa (1605), TDoubleRec.Exponent (1606), TDoubleRec.SpecialType (1606)

75.13.5 TDoubleRec.Exponent

Synopsis: Exponent of the floating point value.

Declaration: `function Exponent : LongInt`

Visibility: public

Description: `Exponent` returns the exponent (`X` in $m*2\hat{X}$ representation) of the floating point value.

See also: TDoubleRec.Fraction (1606), TDoubleRec.Mantissa (1605), TDoubleRec.SpecialType (1606)

75.13.6 TDoubleRec.SpecialType

Synopsis: Is the floating point value special ?

Declaration: `function SpecialType : TFloatSpecial`

Visibility: public

Description: `SpecialType` returns special characteristics of the floating point value, if any. See TFLoatSpecial (1417) for an explanation of the various special values.

See also: TDoubleRec.Fraction (1606), TDoubleRec.Mantissa (1605), TDoubleRec.Exponent (1606), TFLoatSpecial (1417)

75.13.7 TDoubleRec.BuildUp

Synopsis: Build a double value.

Declaration:
```
procedure BuildUp(const _Sign: Boolean; const _Mantissa: QWord;
                  const _Exponent: LongInt)
```

Visibility: public

Description: `Buildup` will build a double value from the given `_Sign`, `_Mantissa` and `_Exponent`.

See also: TDoubleRec.Mantissa (1605), TDoubleRec.Sign (1607), TDoubleRec.Exp (1607)

75.13.8 TDoubleRec.Sign

Synopsis: Sign of the floating point value.

Declaration: `Property Sign : Boolean`

Visibility: public

Access: Read,Write

75.13.9 TDoubleRec.Exp

Synopsis: Exponent bitpattern representation.

Declaration: `Property Exp : QWord`

Visibility: public

Access: Read,Write

Description: `Exp` returns the internal bit representation of the exponent of the floating point value.

See also: TDoubleRec.Sign (1607), TDoubleRec.Exponent (1606)

75.13.10 TDoubleRec.Frac

Synopsis: Fractional part of double.

Declaration: `Property Frac : QWord`

Visibility: public

Access: Read,Write

Description: `Frac` is the fractional part of the Double. This is the bit pattern representing the fractional part including the preceding 1. (The mantissa is that bit pattern without the preceding 1)

See also: Frac (1499)

75.14 TExtended80Rec

```
TExtended80Rec = packed record
private
  Bias = $3FFF;
  function
  GetExp : QWord;
  procedure SetExp(e: QWord);
  function GetSign
  : Boolean;
  procedure SetSign(s: Boolean);
public
  function Mantissa
  (IncludeHiddenBit: Boolean) : QWord;
  function Fraction : Extended
  ;
  function Exponent : LongInt;
```

```
  property Sign : Boolean;
  property
  Exp : QWord;
  function SpecialType : TFloatSpecial;
  procedure
  BuildUp(const _Sign: Boolean; const _Mantissa: QWord;
  const _Exponent: LongInt);
case Byte of
0: (
public
  Bytes
  : Array[0..9] of Byte;
);
1: (
public
  Words : Array[0..4] of Word
  ;
);
2: (
public
  _Exp : Word;
  Frac : QWord;
);
3: (
public
  Value
  : Extended;
);
end
```

`TExtended80Rec` models the memory layout of an extended value when using software floating point math.

75.14.1 Method overview

Page	Method	Description
1609	BuildUp	
1609	Exponent	Exponent of the floating point value.
1609	Fraction	Fraction of the floating point value.
1608	Mantissa	Mantissa of the floating point value.
1609	SpecialType	Is the floating point value special ?

75.14.2 Property overview

Page	Properties	Access	Description
1610	Exp	rw	Exponent representation.
1610	Sign	rw	Sign of the floating point value.

75.14.3 TExtended80Rec.Mantissa

Synopsis: Mantissa of the floating point value.

Declaration: `function Mantissa(IncludeHiddenBit: Boolean) : QWord`

Visibility: public

Description: `Mantissa` returns the Mantissa part (bit pattern of the significand, skipping the hidden bit) of the floating point value.

See also: TExtended80Rec.Fraction (1609), TExtended80Rec.Exponent (1609), TExtended80Rec.SpecialType (1609)

75.14.4 TExtended80Rec.Fraction

Synopsis: Fraction of the floating point value.

Declaration: `function Fraction : Extended`

Visibility: public

Description: `Fraction` returns the fraction (value after decimal) of the floating point value.

See also: TExtended80Rec.Mantissa (1608), TExtended80Rec.Exponent (1609), TExtended80Rec.SpecialType (1609)

75.14.5 TExtended80Rec.Exponent

Synopsis: Exponent of the floating point value.

Declaration: `function Exponent : LongInt`

Visibility: public

Description: `Exponent` returns the exponent (the `X` in the $m*2\hat{X}$ representation) of the floating point value.

See also: TExtended80Rec.Fraction (1609), TExtended80Rec.Mantissa (1608), TExtended80Rec.SpecialType (1609)

75.14.6 TExtended80Rec.SpecialType

Synopsis: Is the floating point value special ?

Declaration: `function SpecialType : TFloatSpecial`

Visibility: public

Description: `SpecialType` returns special characteristics of the floating point value, if any. See TFLoatSpecial (1417) for an explanation of the various special values.

See also: TExtended80Rec.Fraction (1609), TExtended80Rec.Mantissa (1608), TExtended80Rec.Exponent (1609), TFLoatSpecial (1417)

75.14.7 TExtended80Rec.BuildUp

Declaration:
```
procedure BuildUp(const _Sign: Boolean; const _Mantissa: QWord;
                  const _Exponent: LongInt)
```

Visibility: public

75.14.8 TExtended80Rec.Sign

Synopsis: Sign of the floating point value.

Declaration: `Property Sign : Boolean`

Visibility: public

Access: Read,Write

75.14.9 TExtended80Rec.Exp

Synopsis: Exponent representation.

Declaration: `Property Exp : QWord`

Visibility: public

Access: Read,Write

Description: `Exponent` returns the internal bit representation of the exponent of the floating point value.

See also: TExtended80Rec.Sign (1610), TExtended80Rec.Exponent (1609)

75.15 tinterfaceentry

```
tinterfaceentry = record
private
  function GetIID : PGuid;
  function
  GetIIDStr : PShortString;
public
  property IID : PGuid;
  property
  IIDStr : PShortString;
  IIDRef : ^PGuid;
  VTable : Pointer;
case
  Integer of
1: (
public
  IOffset : SizeUInt;
);
2: (
public
  IOffsetAsCodePtr
  : CodePointer;
  IIDStrRef : ^PShortString;
  IType : tinterfaceentrytype
  ;
);
end
```

`tinterfaceentry` is used to store the list of Interfaces of a class. This list is stored as an array of `tinterfaceentry` records.

75.15.1 Property overview

Page	Properties	Access	Description
1611	IID	r	Unique GUID for this interface.
1611	IIDStr	r	Pointer to GUID string. Always assigned for COM.

75.15.2 tinterfaceentry.IID

Synopsis: Unique GUID for this interface.

Declaration: `Property IID : PGuid`

Visibility: public

Access: Read

75.15.3 tinterfaceentry.IIDStr

Synopsis: Pointer to GUID string. Always assigned for COM.

Declaration: `Property IIDStr : PShortString`

Visibility: public

Access: Read

75.16 TSingleRec

```
TSingleRec = packed record
private
  Bias = $7F;
  function GetExp
  : QWord;
  procedure SetExp(e: QWord);
  function GetSign : Boolean
  ;
  procedure SetSign(s: Boolean);
  function GetFrac : QWord;
  procedure
  SetFrac(e: QWord);
public
  function Mantissa(IncludeHiddenBit: Boolean
  ) : QWord;
  function Fraction : ValReal;
  function Exponent : LongInt
  ;
  property Sign : Boolean;
  property Exp : QWord;
  property Frac
  : QWord;
  function SpecialType : TFloatSpecial;
  procedure BuildUp
  (const _Sign: Boolean; const _Mantissa: QWord;
  const _Exponent: LongInt);
```

```
case Byte of
0: (
public
  Bytes : Array
  [0..3] of Byte;
);
1: (
public
  Words : Array[0..1] of Word;
);
2
  : (
public
  Data : DWord;
);
3: (
public
  Value : Single;
);
end
```

TsingleRec models the memory layout of a double value when using software floating point math.

75.16.1 Method overview

Page	Method	Description
1613	BuildUp	
1613	Exponent	Exponent of the floating point value.
1613	Fraction	Fraction of the floating point value.
1612	Mantissa	Mantissa of the floating point value.
1613	SpecialType	Is the floating point value special ?

75.16.2 Property overview

Page	Properties	Access	Description
1614	Exp	rw	Exponent bitpattern representation.
1614	Frac	rw	Fractional part of single.
1613	Sign	rw	Sign of the floating point value.

75.16.3 TSingleRec.Mantissa

Synopsis: Mantissa of the floating point value.

Declaration: `function Mantissa(IncludeHiddenBit: Boolean) : QWord`

Visibility: public

Description: Mantissa returns the Mantissa part (significand bitpattern without hidden bit) of the floating point value.

See also: TsingleRec.Fraction (1613), TsingleRec.Exponent (1613), TsingleRec.SpecialType (1613)

75.16.4 TSingleRec.Fraction

Synopsis: Fraction of the floating point value.

Declaration: `function Fraction : ValReal`

Visibility: public

Description: `Fraction` returns the fraction (after decimal) of the floating point value.

See also: TsingleRec.Mantissa (1612), TsingleRec.Exponent (1613), TsingleRec.SpecialType (1613)

75.16.5 TSingleRec.Exponent

Synopsis: Exponent of the floating point value.

Declaration: `function Exponent : LongInt`

Visibility: public

Description: `Exponent` returns the exponent (`X` in $m*2\hat{X}$ representation) of the floating point value.

See also: TsingleRec.Fraction (1613), TsingleRec.Mantissa (1612), TsingleRec.SpecialType (1613)

75.16.6 TSingleRec.SpecialType

Synopsis: Is the floating point value special ?

Declaration: `function SpecialType : TFloatSpecial`

Visibility: public

Description: `SpecialType` returns special characteristics of the floating point value, if any. See TFLoatSpecial (1417) for an explanation of the various special values.

See also: TsingleRec.Fraction (1613), TsingleRec.Mantissa (1612), TsingleRec.Exponent (1613), TFLoatSpecial (1417)

75.16.7 TSingleRec.BuildUp

Declaration:
```
procedure BuildUp(const _Sign: Boolean; const _Mantissa: QWord;
                  const _Exponent: LongInt)
```

Visibility: public

75.16.8 TSingleRec.Sign

Synopsis: Sign of the floating point value.

Declaration: `Property Sign : Boolean`

Visibility: public

Access: Read,Write

75.16.9 TSingleRec.Exp

Synopsis: Exponent bitpattern representation.

Declaration: `Property Exp : QWord`

Visibility: public

Access: Read,Write

Description: `Exp` returns the internal bit representation of the exponent of the floating point value.

See also: TsingleRec.Sign (1613), TsingleRec.Exponent (1613)

75.16.10 TSingleRec.Frac

Synopsis: Fractional part of single.

Declaration: `Property Frac : QWord`

Visibility: public

Access: Read,Write

Description: `Frac` is the fractional part of the Single. his is the bit pattern representing the fractional part including the preceding 1. (The mantissa is that bit pattern without the preceding 1)

See also: Frac (1499)

75.17 TVmt

```
TVmt = record
public
  vInstanceSize : SizeInt;
  vInstanceSize2
  : SizeInt;
  vParentRef : PPVmt;
  vClassName : PShortString;
  vDynamicTable
  : Pointer;
  vMethodTable : Pointer;
  vFieldTable : Pointer;
  vTypeInfo
  : Pointer;
  vInitTable : Pointer;
  vAutoTable : Pointer;
  vIntfTable
  : pinterfacetable;
  vMsgStrPtr : pstringmessagetable;
  vDestroy
  : CodePointer;
  vNewInstance : CodePointer;
  vFreeInstance : CodePointer
  ;
  vSafeCallException : CodePointer;
  vDefaultHandler : CodePointer
```

```
  ;
  vAfterConstruction : CodePointer;
  vBeforeDestruction : CodePointer
  ;
  vDefaultHandlerStr : CodePointer;
  vDispatch : CodePointer;
  vDispatchStr : CodePointer;
  vEquals : CodePointer;
  vGetHashCode
  : CodePointer;
  vToString : CodePointer;
private
  function GetvParent
  : PVmt;
public
  property vParent : PVmt;
end
```

TVMT is a record describing the VMT of a class. It's various fields represent the available information in the VMT, as far as it is common to all classes.

75.17.1 Property overview

Page	Properties	Access	Description
1615	vParent	r	Pointer to parent VMT.

75.17.2 TVmt.vParent

Synopsis: Pointer to parent VMT.

Declaration: `Property vParent : PVmt`

Visibility: public

Access: Read

75.18 IDispatch

75.18.1 Description

IDispatch is the pascal definition of the Windows Dispatch interface definition.

See also: IUnknown (1619)

75.18.2 Method overview

Page	Method	Description
1616	GetIDsOfNames	Return IDs of named procedures.
1616	GetTypeInfo	Return type information about properties.
1616	GetTypeInfoCount	Return number of properties.
1616	Invoke	Invoke a dispatch method.

75.18.3 IDispatch.GetTypeInfoCount

Synopsis: Return number of properties.

Declaration:
```
function GetTypeInfoCount(out count: LongInt) : HRESULT
```

Visibility: default

75.18.4 IDispatch.GetTypeInfo

Synopsis: Return type information about properties.

Declaration:
```
function GetTypeInfo(Index: LongInt; LocaleID: LongInt; out TypeInfo)
                            : HRESULT
```

Visibility: default

75.18.5 IDispatch.GetIDsOfNames

Synopsis: Return IDs of named procedures.

Declaration:
```
function GetIDsOfNames(const iid: TGuid; names: Pointer;
                             NameCount: LongInt; LocaleID: LongInt;
                             DispIDs: Pointer) : HRESULT
```

Visibility: default

Description: Return the ID of a procedure.

75.18.6 IDispatch.Invoke

Synopsis: Invoke a dispatch method.

Declaration:
```
function Invoke(DispID: LongInt; const iid: TGuid; LocaleID: LongInt;
                       Flags: Word; var params; VarResult: pointer;
                       ExcepInfo: pointer; ArgErr: pointer) : HRESULT
```

Visibility: default

75.19 IEnumerable

75.19.1 Description

`IEnumerable` can be used to get an enumerator from a class. If a class implements `IEnumerable`, it can return an enumerator interface IEnumerator (1617).

See also: IEnumerator (1617)

75.19.2 Method overview

Page	Method	Description
1617	GetEnumerator	Return an enumerator interface for this class.

75.19.3 IEnumerable.GetEnumerator

Synopsis: Return an enumerator interface for this class.

Declaration: `function GetEnumerator : IEnumerator`

Visibility: default

Description: `GetEnumerator` returns a new IEnumerator (1617) interface for this class. This is called by the compiler whenever a `for in` loop is encountered in the source code to retrieve the enumerator instance.

See also: IEnumerator (1617)

75.20 IEnumerator

75.20.1 Description

`IEnumerator` is the interface needed by the `For ... in ...` language construct, when operating on classes. It contains all methods that the compiler needs to implement a loop.

A `for in` loop like the following:

```
For O in MyObject do
  begin
  // do things
  end;
```

is treated by the compiler as equivalent to the following code:

```
Var
  I : IEnumerator;
  O : TObject;

begin
  I:=MyObject.GetEnumerator;
  While I.MoveNext do
    begin
    O:=I.GetCurrent;
    // Do things
    end;
end.
```

Any class that implements the `IEnumerable` interface must be able to return an `IEnumerator` instance for the compiler to use in a `For in` loop.

See also: IEnumerable (1616)

75.20.2 Method overview

Page	Method	Description
1618	GetCurrent	Returns the current element in the iteration cycle.
1618	MoveNext	Move to the next value.
1618	Reset	Reset the pointer.

75.20.3 Property overview

Page	Properties	Access	Description
1618	Current	r	Return the current item.

75.20.4 IEnumerator.GetCurrent

Synopsis: Returns the current element in the iteration cycle.

Declaration: `function GetCurrent : TObject`

Visibility: default

Description: `getCurrent` should return the object instance representing the current value in the `for in` loop. `GetCurrent` will always be called immediately after IEnumerator.MoveNext (1618) returned `True`.

Remark The actual return type of the interface should not necessarily be `TObject`, it can be any type. The compiler will check the actual return type with the type of the loop variable, and they should match.

See also: IEnumerator.MoveNext (1618), IEnumerator.Reset (1618)

75.20.5 IEnumerator.MoveNext

Synopsis: Move to the next value.

Declaration: `function MoveNext : Boolean`

Visibility: default

Description: `MoveNext` should move the current item pointer to the next available item. It should return `True` if an item is available, `False` if no more items are available. The first time it is called It will be called at the beginning of the for loop, so it should position the enumerator on the first value (if there is one). After `MoveNext` has returned `True`, IEnumerator.GetCurrent (1618) will be called to retrieve the item.

See also: IEnumerator.Reset (1618), IEnumerator.GetCurrent (1618)

75.20.6 IEnumerator.Reset

Synopsis: Reset the pointer.

Declaration: `procedure Reset`

Visibility: default

Description: `Reset` can be implemented to put the pointer at the start of the list. It is not mandatory to implement this method, the compiler does not use it.

See also: IEnumerator.GetCurrent (1618), IEnumerator.MoveNext (1618)

75.20.7 IEnumerator.Current

Synopsis: Return the current item.

Declaration: `Property Current : TObject`

Visibility: default

Access: Read

Description: `Current` simply is the redefinition of IEnumerator.GetCurrent (1618) as a property. It is read-only.

See also: IEnumerator.GetCurrent (1618), IEnumerator.MoveNext (1618)

75.21 IInvokable

75.21.1 Description

`IInvokable` is a descendent of IInterface (1399), compiled in the `{$M+}` state, so Run-Time Type Information (RTTI) is generated for it.

See also: IDispatch (1615), IInterface (1399)

75.22 IUnknown

75.22.1 Description

`IUnknown` is defined by windows. It's the basic interface which all COM objects must implement. The definition does not contain any code.

See also: IInterface (1399), IDispatch (1615), IInvokable (1619)

75.22.2 Method overview

Page	Method	Description
1619	QueryInterface	Return pointer to VMT table of interface.
1619	_AddRef	Increase reference count of the interface.
1620	_Release	Decrease reference count of the interface.

75.22.3 IUnknown.QueryInterface

Synopsis: Return pointer to VMT table of interface.

Declaration: `function QueryInterface(const iid: TGuid; out obj) : LongInt`

Visibility: default

75.22.4 IUnknown._AddRef

Synopsis: Increase reference count of the interface.

Declaration: `function _AddRef : LongInt`

Visibility: default

See also: IUnknown._Release (1620)

75.22.5 IUnknown._Release

Synopsis: Decrease reference count of the interface.

Declaration: `function _Release : LongInt`

Visibility: default

See also: IUnknown._AddRef (1619)

75.23 TAggregatedObject

75.23.1 Description

`TAggregatedObject` implements an object whose lifetime is governed by an external object (or interface). It does not implement the `IUnknown` interface by itself, but delegates all methods to the controller object, as exposed in the Controller (1621) property. In effect, the reference count of the aggregated object is the same as that of it's controller, and additionally, all interfaces of the controller are exposed by the aggregated object.

Note that the aggregated object maintains a non-counted reference to the controller.

Aggregated objects should be used when using delegation to implement reference counted objects: the delegated interfaces can be implemented safely by `TAggregatedObject` descendents.

See also: Create (1620), Controller (1621)

75.23.2 Method overview

Page	Method	Description
1620	Create	Create a new instance of `TAggregatedObject`.

75.23.3 Property overview

Page	Properties	Access	Description
1621	Controller	r	Controlling instance.

75.23.4 TAggregatedObject.Create

Synopsis: Create a new instance of `TAggregatedObject`.

Declaration: `constructor Create(const aController: IUnknown)`

Visibility: public

Description: `Create` creates a new instance of `TAggregatedObject` on the heap, and stores a reference to `aController`, so it can be exposed in the Controller (1621) property.

Errors: If not enough memory is present on the heap, an exception will be raised. If the `aController` is `Nil`, exceptions will occur when any of the `TAggregatedObject` methods (actually, the `IUnknown` methods) are used.

See also: Controller (1621)

75.23.5 TAggregatedObject.Controller

Synopsis: Controlling instance.

Declaration: `Property Controller : IUnknown`

Visibility: public

Access: Read

Description: `Controller` exposes the controlling object, with all interfaces it has.

The value of the controller is set when the `TAggregatedObject` instance is created.

See also: TAggregatedObject.Create (1620)

75.24 TContainedObject

75.24.1 Description

`TContainedObject` is the base class for contained objects, i.e. objects that do not implement a reference counting mechanism themselves, but are owned by some other object which handles the reference counting mechanism. It implements the `IUnknown` interface and, more specifically, the `QueryInterface` method of `IUnknown`.

See also: IInterface (1399)

75.24.2 Interfaces overview

Page	Interfaces	Description
1399	IInterface	Basic interface for all COM based interfaces.

75.25 TInterfacedObject

75.25.1 Description

`TInterfacedObject` is a descendent of TObject (1623) which implements the IUnknown (1619) interface. It can be used as a base class for all classes which need reference counting.

See also: IUnknown (1619), TObject (1623)

75.25.2 Interfaces overview

Page	Interfaces	Description
1619	IUnknown	Basic interface for all COM-based interfaces.

75.25.3 Method overview

Page	Method	Description
1622	AfterConstruction	Handle reference count properly.
1622	BeforeDestruction	Check reference count.
1622	destroy	Destroy interfaced object.
1622	NewInstance	Create a new instance.

75.25.4 Property overview

Page	Properties	Access	Description
1623	RefCount	r	Return the current reference count.

75.25.5 TInterfacedObject.destroy

Synopsis: Destroy interfaced object.

Declaration: `destructor destroy;  Override`

Visibility: public

Description: `Destroy` overrides the default destructor to clean up: it explicitly sets all counters to zero.

See also: IUnknown._Release (1620)

75.25.6 TInterfacedObject.AfterConstruction

Synopsis: Handle reference count properly.

Declaration: `procedure AfterConstruction;  Override`

Visibility: public

Description: `AfterConstruction` overrides the basic method in `TObject` and adds some additional reference count handling.

Errors: None.

See also: BeforeDestruction (1622)

75.25.7 TInterfacedObject.BeforeDestruction

Synopsis: Check reference count.

Declaration: `procedure BeforeDestruction;  Override`

Visibility: public

Description: `AfterConstruction` overrides the basic method in `TObject` and adds a reference count check: if the reference count is not zero, an error occurs.

Errors: A runtime-error 204 will be generated if the reference count is nonzero when the object is destroyed.

See also: AfterConstruction (1622)

75.25.8 TInterfacedObject.NewInstance

Synopsis: Create a new instance.

Declaration: `class function NewInstance : TObject;  Override`

Visibility: public

Description: `NewInstance` initializes a new instance of TInterfacedObject (1621)

Errors: None.

75.25.9 TInterfacedObject.RefCount

Synopsis: Return the current reference count.

Declaration: `Property RefCount : LongInt`

Visibility: public

Access: Read

Description: `RefCount` returns the current reference count. This reference count cannot be manipulated, except through the methods of IUnknown (1619). When it reaches zero, the class instance is destroyed.

See also: IUnknown (1619)

75.26 TObject

75.26.1 Description

`TObject` is the parent root class for all classes in Object Pascal. If a class has no parent class explicitly declared, it is dependent on `TObject`. `TObject` introduces class methods that deal with the class' type information, and contains all necessary methods to create an instance at runtime, and to dispatch messages to the correct method (both string and integer messages).

See also: TClass (1412)

75.26.2 Method overview

Page	Method	Description
1630	AfterConstruction	Method called after the constructor was called.
1630	BeforeDestruction	Method called before the destructor is called.
1627	ClassInfo	Return a pointer to the type information for this class.
1627	ClassName	Return the current class name.
1628	ClassNameIs	Check whether the class name equals the given name.
1628	ClassParent	Return the parent class.
1627	ClassType	Return a "class of" pointer for the current class.
1627	CleanupInstance	Finalize the class instance.
1624	Create	`TObject` Constructor.
1626	DefaultHandler	Default handler for integer message handlers.
1630	DefaultHandlerStr	Default handler for string messages.
1625	Destroy	`TObject` destructor.
1630	Dispatch	Dispatch an integer message.
1631	DispatchStr	Dispatch a string message.
1633	Equals	Check if two objects are equal.
1629	FieldAddress	Return the address of a field.
1626	Free	Check for `Nil` and call destructor.
1625	FreeInstance	Clean up instance and free the memory reserved for the instance.
1633	GetHashCode	Return a hash code for the object.
1631	GetInterface	Return a reference to an interface.
1631	GetInterfaceByStr	Return an interface based on its GUID.
1632	GetInterfaceEntry	Return the interface table entry by GUID.
1632	GetInterfaceEntryByStr	Return the interface table entry by string.
1632	GetInterfaceTable	Return a pointer to the table of implemented interfaces for a class.
1632	GetInterfaceWeak	Get a reference to an interface, not increasing the reference count.
1628	InheritsFrom	Check whether class is an ancestor.
1626	InitInstance	Initialize a new class instance.
1628	InstanceSize	Return the size of an instance.
1629	MethodAddress	Return the address of a method.
1629	MethodName	Return the name of a method.
1625	newinstance	Allocate memory on the heap for a new instance.
1633	QualifiedClassName	Fully qualified classname.
1626	SafeCallException	Handle exception object.
1629	StringMessageTable	Return a pointer to the string message table.
1634	ToString	Return a string representation for the object.
1633	UnitName	Unit name.

75.26.3 TObject.Create

Synopsis: `TObject` Constructor.

Declaration: `constructor Create`

Visibility: public

Description: `Create` creates a new instance of `TObject`. Currently it does nothing (except allocating memory for the new instance).

Note that allocating the memory for the object instance happens in hidden code generated as part of every constructor, not only in `TObject.Create`. The hidden code calls NewInstance (1625).

See also: Destroy (1625)

75.26.4 TObject.Destroy

Synopsis: `TObject` destructor.

Declaration: `destructor Destroy;  Virtual`

Visibility: public

Description: `Destroy` is the destructor of `TObject`. It will clean up the memory assigned to the instance. Descendent classes should override destroy if they want to do additional clean-up. No other destructor should be implemented.

It is bad programming practice to call `Destroy` directly. It is better to call the Free (1626) method, because that one will check first if `Self` is different from `Nil`.

To clean up an instance and reset the reference to the instance, it is best to use the FreeAndNil (1745) function.

See also: Create (1624), Free (1626)

75.26.5 TObject.newinstance

Synopsis: Allocate memory on the heap for a new instance.

Declaration: `class function newinstance : TObject;  Virtual`

Visibility: public

Description: `NewInstance` allocates memory on the heap for a new instance of the current class. If the memory was allocated, the class will be initialized by a call to InitInstance (1626). The function returns the newly initialized instance.

This method can be overridden to implement e.g. object pooling. Note that the method is responsible for correctly initializing the object, i.e. zeroing out the new instance.

Errors: If not enough memory is available, a `Nil` pointer may be returned, or an exception may be raised.

See also: Create (1624), InitInstance (1626), InstanceSize (1628), FreeInstance (1625)

75.26.6 TObject.FreeInstance

Synopsis: Clean up instance and free the memory reserved for the instance.

Declaration: `procedure FreeInstance;  Virtual`

Visibility: public

Description: `FreeInstance` cleans up an instance of the current class, and releases the heap memory occupied by the class instance.

See also: Destroy (1625), InitInstance (1626), NewInstance (1625)

75.26.7 TObject.SafeCallException

Synopsis: Handle exception object.

Declaration:
```
function SafeCallException(exceptobject: TObject;
                           exceptaddr: CodePointer) : HRESULT;  Virtual
```

Visibility: public

Description: `SafeCallException` should be overridden to handle exceptions in a method marked with the savecall directive. The implementation in `TObject` simply returns zero.

75.26.8 TObject.DefaultHandler

Synopsis: Default handler for integer message handlers.

Declaration:
```
procedure DefaultHandler(var message);  Virtual
```

Visibility: public

Description: `DefaultHandler` is the default handler for messages. If a message has an unknown message ID (i.e. does not appear in the table with integer message handlers), then it will be passed to `DefaultHandler` by the Dispatch (1630) method.

See also: Dispatch (1630), DefaultHandlerStr (1630)

75.26.9 TObject.Free

Synopsis: Check for `Nil` and call destructor.

Declaration:
```
procedure Free
```

Visibility: public

Description: `Free` will check the `Self` pointer and calls Destroy (1625) if it is different from `Nil`. This is a safer method than calling `Destroy` directly. If a reference to the object must be reset as well (a recommended technique), then the function FreeAndNil (1745) should be called.

Errors: None.

See also: Destroy (1625), FreeAndNil (1745)

75.26.10 TObject.InitInstance

Synopsis: Initialize a new class instance.

Declaration:
```
class function InitInstance(instance: pointer) : TObject
```

Visibility: public

Description: `InitInstance` initializes the memory pointer to by `Instance`. This means that the VMT is initialized, and the interface pointers are set up correctly. The function returns the newly initialized instance.

See also: NewInstance (1625), Create (1624)

75.26.11 TObject.CleanupInstance

Synopsis: Finalize the class instance.

Declaration: `procedure CleanupInstance`

Visibility: public

Description: `CleanUpinstance` finalizes the instance, i.e. takes care of all reference counted objects, by decreasing their reference count by 1, and freeing them if their count reaches zero.

Normally, `CleanupInstance` should never be called, it is called automatically when the object is freed with it's constructor.

Errors: None.

See also: Destroy (1625), Free (1626), InitInstance (1626)

75.26.12 TObject.ClassType

Synopsis: Return a "class of" pointer for the current class.

Declaration: `class function ClassType : TClass`

Visibility: public

Description: `ClassType` returns a TClass (1412) class type reference for the current class.

See also: TClass (1412), ClassInfo (1627), ClassName (1627)

75.26.13 TObject.ClassInfo

Synopsis: Return a pointer to the type information for this class.

Declaration: `class function ClassInfo : pointer`

Visibility: public

Description: `ClassInfo` returns a pointer to the type information for this class. This pointer can be used in the various type information routines.

75.26.14 TObject.ClassName

Synopsis: Return the current class name.

Declaration: `class function ClassName : shortstring`

Visibility: public

Description: `ClassName` returns the class name for the current class. To check for the class name, you can use the ClassNameIs (1628) class method.

Errors: None.

See also: ClassInfo (1627), ClassType (1627), ClassNameIs (1628)

75.26.15 TObject.ClassNameIs

Synopsis: Check whether the class name equals the given name.

Declaration: `class function ClassNameIs(const name: string) : Boolean`

Visibility: public

Description: `ClassNameIs` checks whether `Name` equals the class name. It takes of case sensitivity.

See also: ClassInfo (1627), ClassType (1627), ClassName (1627)

75.26.16 TObject.ClassParent

Synopsis: Return the parent class.

Declaration: `class function ClassParent : TClass`

Visibility: public

Description: `ClassParent` returns the class of the parent class of the current class. This is always different from `Nil`, except for `TObject`.

Errors: None.

See also: ClassInfo (1627), ClassType (1627), ClassNameIs (1627)

75.26.17 TObject.InstanceSize

Synopsis: Return the size of an instance.

Declaration: `class function InstanceSize : SizeInt`

Visibility: public

Description: `InstanceSize` returns the number of bytes an instance takes in memory. This is Just the memory occupied by the class structure, and does not take into account any additional memory that might be allocated by the constructor of the class.

Errors: None.

See also: InitInstance (1626), ClassName (1627), ClassInfo (1627), ClassType (1627)

75.26.18 TObject.InheritsFrom

Synopsis: Check whether class is an ancestor.

Declaration: `class function InheritsFrom(aclass: TClass) : Boolean`

Visibility: public

Description: `InheritsFrom` returns `True` if `AClass` is an ancestor class from the current class, and returns false if it is not.

See also: ClassName (1627), ClassInfo (1627), ClassType (1627), TClass (1412)

75.26.19 TObject.StringMessageTable

Synopsis: Return a pointer to the string message table.

Declaration: `class function StringMessageTable : pstringmessagetable`

Visibility: public

Description: `StringMessageTable` returns a pointer to the string message table, which can be used to look up methods for dispatching a string message. It is used by the DispatchStr (1631) method.

Errors: If there are no string message handlers, nil is returned.

See also: DispatchStr (1631), Dispatch (1630)

75.26.20 TObject.MethodAddress

Synopsis: Return the address of a method.

Declaration: `class function MethodAddress(const name: shortstring) : CodePointer`

Visibility: public

Description: `MethodAddress` returns the address of a method, searching the method by its name. The `Name` parameter specifies which method should be taken. The search is conducted in a case-insensitive manner.

Errors: If no matching method is found, `Nil` is returned.

See also: MethodName (1629), FieldAddress (1629)

75.26.21 TObject.MethodName

Synopsis: Return the name of a method.

Declaration: `class function MethodName(address: CodePointer) : shortstring`

Visibility: public

Description: `MethodName` searches the VMT for a method with the specified `address` and returns the name of the method.

Errors: If no method with the matching address is found, an empty string is returned.

See also: MethodAddress (1629), FieldAddress (1629)

75.26.22 TObject.FieldAddress

Synopsis: Return the address of a field.

Declaration: `function FieldAddress(const name: shortstring) : pointer`

Visibility: public

Description: `FieldAddress` returns the address of the field with name `name`. The address is the address of the field in the current class instance.

Errors: If no field with the specified name is found, `Nil` is returned.

See also: MethodAddress (1629), MethodName (1629)

75.26.23 TObject.AfterConstruction

Synopsis: Method called after the constructor was called.

Declaration: `procedure AfterConstruction; Virtual`

Visibility: public

Description: `AfterConstruction` is a method called after the constructor was called. It does nothing in the implementation of `TObject` and must be overridden by descendent classes to provide specific behaviour that is executed after the constructor has finished executing. (for instance, call an event handler)

Errors: None.

See also: BeforeDestruction (1630), Create (1624)

75.26.24 TObject.BeforeDestruction

Synopsis: Method called before the destructor is called.

Declaration: `procedure BeforeDestruction; Virtual`

Visibility: public

Description: `BeforeDestruction` is a method called before the destructor is called. It does nothing in the implementation of `TObject` and must be overridden by descendent classes to provide specific behaviour that is executed before the destructor has finished executing. (for instance, call an event handler)

Errors: None.

See also: AfterConstruction (1630), Destroy (1625), Free (1626)

75.26.25 TObject.DefaultHandlerStr

Synopsis: Default handler for string messages.

Declaration: `procedure DefaultHandlerStr(var message); Virtual`

Visibility: public

Description: `DefaultHandlerStr` is called for string messages which have no handler associated with them in the string message handler table. The implementation of `DefaultHandlerStr` in `TObject` does nothing and must be overridden by descendent classes to provide specific message handling behaviour.

See also: DispatchStr (1631), Dispatch (1630), DefaultHandler (1626)

75.26.26 TObject.Dispatch

Synopsis: Dispatch an integer message.

Declaration: `procedure Dispatch(var message); Virtual`

Visibility: public

Description: `Dispatch` looks in the message handler table for a handler that handles `message`. The message is identified by the first dword (cardinal) in the message structure.

If no matching message handler is found, the message is passed to the DefaultHandler (1626) method, which can be overridden by descendent classes to add custom handling of messages.

See also: DispatchStr (1631), DefaultHandler (1626)

75.26.27 TObject.DispatchStr

Synopsis: Dispatch a string message.

Declaration:
```
procedure DispatchStr(var message);  Virtual
```

Visibility: public

Description: `DispatchStr` extracts the message identifier from `Message` and checks the message handler table to see if a handler for the message is found, and calls the handler, passing along the message. If no handler is found, the default DefaultHandlerStr (1630) is called.

Errors: None.

See also: DefaultHandlerStr (1630), Dispatch (1630), DefaultHandler (1626)

75.26.28 TObject.GetInterface

Synopsis: Return a reference to an interface.

Declaration:
```
function GetInterface(const iid: TGuid; out obj) : Boolean
function GetInterface(const iidstr: shortstring; out obj) : Boolean
```

Visibility: public

Description: `GetInterface` scans the interface tables and returns a reference to the interface `iid`. The reference is stored in `Obj` which should be an interface reference. It returns `True` if the interface was found, `False` if not.

The reference count of the interface is increased by this call.

Errors: If no interface was found, `False` is returned.

See also: GetInterfaceByStr (1631)

75.26.29 TObject.GetInterfaceByStr

Synopsis: Return an interface based on its GUID.

Declaration:
```
function GetInterfaceByStr(const iidstr: shortstring; out obj) : Boolean
```

Visibility: public

Description: `GetInterfaceByStr` returns in `obj` a pointer to the interface identified by `iidstr`. The function returns `True` if the interface is indeed implemented by the class, or `False` otherwise.

The `iidstr` is the unique GUID by which the interface was declared.

Errors: The function returns false if the requested interface is not implemented.

See also: TObject.GetInterfaceEntry (1632), TObject.GetInterfaceEntryByStr (1632)

75.26.30 TObject.GetInterfaceWeak

Synopsis: Get a reference to an interface, not increasing the reference count.

Declaration:
```
function GetInterfaceWeak(const iid: TGuid; out obj) : Boolean
```

Visibility: public

Description: `GetInterfaceWeak` performs the same function as Getinterface (1631), but unlike the latter, it will not increase the reference count of the interface.

See also: TObject.GetInterface (1631)

75.26.31 TObject.GetInterfaceEntry

Synopsis: Return the interface table entry by GUID.

Declaration:
```
class function GetInterfaceEntry(const iid: TGuid) : pinterfaceentry
```

Visibility: public

Description: `GetInterfaceEntry` returns the internal interface table entry for the interface identified by `iid` (the GUID used in the declaration of the interface). If the interface is not implemented by the class, the function returns `Nil`.

See also: TObject.GetInterfaceByStr (1631), TObject.GetInterfaceEntryByStr (1632)

75.26.32 TObject.GetInterfaceEntryByStr

Synopsis: Return the interface table entry by string.

Declaration:
```
class function GetInterfaceEntryByStr(const iidstr: shortstring)
                                      : pinterfaceentry
```

Visibility: public

Description: `GetInterfaceEntryByStr` returns the internal interface table entry for the interface identified by `iidstr` (A string representation of the GUID used in the declaration of the interface). If the interface is not implemented by the class, the function returns `Nil`.

See also: TObject.GetInterfaceByStr (1631), TObject.GetInterfaceEntry (1632)

75.26.33 TObject.GetInterfaceTable

Synopsis: Return a pointer to the table of implemented interfaces for a class.

Declaration:
```
class function GetInterfaceTable : pinterfacetable
```

Visibility: public

Description: `GetInterfaceTable` returns a pointer to the internal table of implemented interfaces for a class. The result will always point to a valid address, if the class implements no interfaces the `EntryCount` field of the interface table will be zero.

See also: TObject.GetInterfaceByStr (1631), TObject.GetInterfaceEntry (1632)

75.26.34 TObject.UnitName

Synopsis: Unit name.

Declaration: `class function UnitName : ansistring`

Visibility: public

Description: `UnitName` returns the unit name in which the class was defined. The name is obtained from the class definition data the compiler generates for each class.

75.26.35 TObject.QualifiedClassName

Synopsis: Fully qualified classname.

Declaration: `class function QualifiedClassName : ansistring`

Visibility: public

Description: `QualifiedClassName` is the classname TObject.ClassName (1627) prepended with the unit name TObject.UnitName (1633) of the unit in which the class is defined. As such, this is a unique name for each class in the program.

See also: TObject.ClassName (1627), TObject.UnitName (1633)

75.26.36 TObject.Equals

Synopsis: Check if two objects are equal.

Declaration: `function Equals(Obj: TObject) : Boolean;  Virtual`

Visibility: public

Description: `Equals` returns `True` if the object instance pointer (`Self`) equals the instance pointer `Obj`.

Descendent classes can override to check properties etc. in case the instance pointers are different.

See also: TObject.GetHashCode (1633), TObject.ToString (1634)

75.26.37 TObject.GetHashCode

Synopsis: Return a hash code for the object.

Declaration: `function GetHashCode : PtrInt;  Virtual`

Visibility: public

Description: `GetHashCode` should return a hash code for the object. By default, the numerical (integer) address of `Self` is returned.

Descendent classes can use this to generate better suitable values to be used in a hash table.

See also: TObject.ToString (1634), TObject.Equals (1633)

75.26.38 TObject.ToString

Synopsis: Return a string representation for the object.

Declaration: `function ToString : ansistring; Virtual`

Visibility: public

Description: `ToString` returns by default the class name of the object. It is useful during sending of debug messages.

Descendent classes can override this method to give a better description of the object than just the class name.

See also: TObject.GetHashCode (1633), TObject.Equals (1633)

Chapter 76

Reference for unit 'sysutils'

76.1 Used units

Table 76.1: Used units by unit 'sysutils'

Name	Page
BaseUnix	146
errors	740
Linux	975
sysconst	??
System	1362
Unix	2135
unixtype	2175

76.2 Overview

This documentation describes the sysutils unit. The sysutils unit was started by Gertjan Schouten, and completed by Michael Van Canneyt. It aims to be compatible to the Delphi sysutils unit, but in contrast with the latter, it is designed to work on multiple platforms. It is implemented on all supported platforms.

76.3 Type Helpers for basic types.

The sysutils unit contains type helpers for basic language types: boolean, ordinals (shortint, smallint, byte, word, integer, cardinal, int64, wqord, nativeint, nativeuint), strings, floating-point and GUID values. They can be used as-is, or they can be descended from to form your own type helpers.

See also: TGuidHelper (1882), TStringHelper (1926), TByteHelper (1855), TShortIntHelper (1910), TSmallIntHelper (1922), TWordHelper (1960), TCardinalHelper (1859), TIntegerHelper (1888), TInt64Helper (1884), TQWordHelper (1906), TNativeIntHelper (1898), TBooleanHelper (1852), TByteBoolHelper (1853), TWordBoolHelper (1959), TLongBoolHelper (1892), TSingleHelper (1916), TDoubleHelper (1863), TExtendedHelper (1876)

76.4 Localization support.

Localization support depends on various constants and structures being initialized correctly. On Windows and OS/2 this is done automatically: a widestring manager is installed by default which helps taking care of the current locale when performing various operations on strings. The various internationalization settings (date/time format, currency, language etc) are also initialized correctly on these platforms.

On Unixes, the widestring support is in a separate unit: cwstring, which loads the various needed functions from the C library. It should be added manually to the uses clause of your program. No internationalization (or localisation) settings are applied by this unit, these must be initialized separately by including the clocale unit in the uses clause of your program.

76.5 Unicode and codepage awareness.

The many functions that deal with filenames in the sysutils routines have been changed from `AnsiString` to RawByteString so they do not perform implicit codepage conversions to the ANSI code page. At the same time, overloaded versions that accept a Unicode string have been created.

For routines that access actual OS functions using single-byte string APIS, the strings are converted to ensure that the OS routine receives a string with the correct encoding when using single-byte strings. This encoding is normally the DefaultFileSystemCodePage (1442) encoding.

On systems with a Unicode I/O API (2-byte strings), the native API is used, meaning that Unicode strings will be passed on as-is, but single-byte strings will be converted (implicitly) to Unicode.

The following is a minimal list of functions that have been changed and duplicated:

Table 76.2:

Name	Description
FileCreate (1715)	Create a new file and return a handle to it.
FileOpen (1721)	Open an existing file and return a file handle.
FileExists (1717)	Check whether a particular file exists in the file system.
DirectoryExists (1701)	Check whether a directory exists in the file system.
FileSetDate (1725)	Set the date of a file.
FileGetAttr (1718)	Return attributes of a file.
FileSetAttr (1724)	Set the attributes of a file.
DeleteFile (1700)	Delete a file from the file system.
RenameFile (1767)	Rename a file.
FileSearch (1722)	Search for a file in a path.
ExeSearch (1708)	Search for an executable.
FindFirst (1727)	Start a file search and return a findhandle.
FindNext (1728)	Find the next entry in a findhandle.
FindClose (1726)	Close a find handle.
FileIsReadOnly (1720)	Check whether a file is read-only.
GetCurrentDir (1747)	Return the current working directory of the application.
SetCurrentDir (1770)	Set the current directory of the application.

The following functions do not interact with the OS, but may nevertheless change the codepage of the strings involved in their operation:

Table 76.3:

Name	Description
ChangeFileExt (1687)	Change the extension of a filename.
ExtractFilePath (1712)	Extract the path from a filename.
ExtractFileDrive (1711)	Extract the drive part from a filename.
ExtractFileName (1712)	Extract the filename part from a full path filename.
ExtractFileExt (1711)	Return the extension from a filename.
ExtractFileDir (1710)	Extract the drive and directory part of a filename.
ExtractShortPathName (1713)	Returns a 8.3 path name.
ExpandFileName (1708)	Expand a relative filename to an absolute filename.
ExpandFileNameCase (1709)	Expand a filename entered as case insensitive to the full path as stored on the disk.
ExtractRelativepath (1712)	Extract a relative path from a filename, given a base directory.
ExpandUNCFileName (1710)	Expand a relative filename to an absolute UNC filename.
IncludeTrailingPathDelimiter (1756)	Add trailing directory separator to a pathname, if needed.
IncludeTrailingBackslash (1756)	Add trailing directory separator to a pathname, if needed.
ExcludeTrailingBackslash (1707)	Strip trailing directory separator from a pathname, if needed.
ExcludeTrailingPathDelimiter (1707)	Strip trailing directory separator from a pathname, if needed.
IncludeLeadingPathDelimiter (1755)	Prepend a path delimiter if there is not already one.
ExcludeLeadingPathDelimiter (1706)	Strip the leading path delimiter of a path.
IsPathDelimiter (1761)	Is the character at the given position a pathdelimiter ?
DoDirSeparators (1703)	Convert known directory separators to the current directory separator.
SetDirSeparators (1770)	Set the directory separators to the known directory separators.
GetDirs (1748)	Return a list of directory names from a path.
ConcatPaths (1691)	Concatenate an array of paths to form a single path.
GetEnvironmentVariable (1749)	Return the value of an environment variable.

76.6 Miscellaneous conversion routines.

Functions for various conversions.

Table 76.4:

Name	Description
BCDToInt (1684)	Convert BCD number to integer
CompareMem (1689)	Compare two memory regions
FloatToStrF (1731)	Convert float to formatted string
FloatToStr (1729)	Convert float to string
FloatToText (1732)	Convert float to string
FormatFloat (1743)	Format a floating point value
GetDirs (1748)	Split string in list of directories
IntToHex (1757)	return hexadecimal representation of integer
IntToStr (1758)	return decimal representation of integer
StrToIntDef (1794)	Convert string to integer with default value
StrToInt (1793)	Convert string to integer
StrToFloat (1791)	Convert string to float
TextToFloat (1800)	Convert null-terminated string to float

76.7 Date/time routines.

Functions for date and time handling.

Table 76.5:

Name	Description
DateTimeToFileDate (1695)	Convert DateTime type to file date
DateTimeToStr (1695)	Construct string representation of DateTime
DateTimeToString (1696)	Construct string representation of DateTime
DateTimeToSystemTime (1697)	Convert DateTime to system time
DateTimeToTimeStamp (1697)	Convert DateTime to timestamp
DateToStr (1698)	Construct string representation of date
Date (1694)	Get current date
DayOfWeek (1699)	Get day of week
DecodeDate (1699)	Decode DateTime to year month and day
DecodeTime (1700)	Decode DateTime to hours, minutes and seconds
EncodeDate (1704)	Encode year, day and month to DateTime
EncodeTime (1704)	Encode hours, minutes and seconds to DateTime
FormatDateTime (1743)	Return string representation of DateTime
IncMonth (1756)	Add 1 to month
IsLeapYear (1760)	Determine if year is leap year
MSecsToTimeStamp (1764)	Convert nr of milliseconds to timestamp
Now (1765)	Get current date and time
StrToDateTime (1790)	Convert string to DateTime
StrToDate (1788)	Convert string to date
StrToTime (1795)	Convert string to time
SystemTimeToDateTime (1799)	Convert system time to datetime
TimeStampToDateTime (1801)	Convert time stamp to DateTime
TimeStampToMSecs (1802)	Convert Timestamp to number of milliseconds
TimeToStr (1802)	return string representation of Time
Time (1801)	Get current time

76.8 File Name handling routines.

Functions for file manipulation.

Table 76.6:

Name	Description
AnsiCompareFileName (1670)	Compare 2 filenames
AnsiLowerCaseFileName (1674)	Create lowercase filename
AnsiUpperCaseFileName (1682)	Create uppercase filename
AddDisk (1668)	Add disk to list of disk drives
ChangeFileExt (1687)	Change extension of file name
CreateDir (1692)	Create a directory
DeleteFile (1700)	Delete a file
DiskFree (1701)	Free space on disk
DiskSize (1702)	Total size of disk
ExpandFileName (1708)	Create full file name
ExpandFileNameCase (1709)	Create full file name case insensitively
ExpandUNCFileName (1710)	Create full UNC file name
ExtractFileDir (1710)	Extract drive and directory part of filename
ExtractFileDrive (1711)	Extract drive part of filename
ExtractFileExt (1711)	Extract extension part of filename
ExtractFileName (1712)	Extract name part of filename
ExtractFilePath (1712)	Extract path part of filename
ExtractRelativePath (1712)	Construct relative path between two files
FileAge (1714)	Return file age
FileDateToDateTime (1716)	Convert file date to system date
FileExists (1717)	Determine whether a file exists on disk
FileGetAttr (1718)	Get attributes of file
FileGetDate (1719)	Get date of last file modification
FileSearch (1722)	Search for file in path
FileSetAttr (1724)	Get file attributes
FileSetDate (1725)	Get file dates
FindFirst (1727)	Start finding a file
FindNext (1728)	Find next file
GetCurrentDir (1747)	Return current working directory
RemoveDir (1766)	Remove a directory from disk
RenameFile (1767)	Rename a file on disk
SameFileName (1769)	Check whether 2 filenames are the same
SetCurrentDir (1770)	Set current working directory
SetDirSeparators (1770)	Set directory separator characters
FindClose (1726)	Stop searching a file
DoDirSeparators (1703)	Replace directory separator characters

76.9 File input/output routines.

Functions for reading/writing to file.

Table 76.7:

Name	Description
FileCreate (1715)	Create a file and return handle
FileOpen (1721)	Open file end return handle
FileRead (1722)	Read from file
FileSeek (1723)	Set file position
FileTruncate (1725)	Truncate file length
FileWrite (1726)	Write to file
FileClose (1715)	Close file handle

76.10 PChar related functions.

Most PChar functions are the same as their counterparts in the STRINGS unit. The following functions are the same :

1. StrCat (1773) : Concatenates two `PChar` strings.
2. StrComp (1774) : Compares two `PChar` strings.
3. StrCopy (1774) : Copies a `PChar` string.
4. StrECopy (1775) : Copies a `PChar` string and returns a pointer to the terminating null byte.
5. StrEnd (1776) : Returns a pointer to the terminating null byte.
6. StrIComp (1777) : Case insensitive compare of 2 `PChar` strings.
7. StrLCat (1779) : Appends at most L characters from one `PChar` to another `PChar`.
8. StrLComp (1780) : Case sensitive compare of at most L characters of 2 `PChar` strings.
9. StrLCopy (1780) : Copies at most L characters from one `PChar` to another.
10. StrLen (1781) : Returns the length (exclusive terminating null byte) of a `PChar` string.
11. StrLIComp (1782) : Case insensitive compare of at most L characters of 2 `PChar` strings.
12. StrLower (1783) : Converts a `PChar` to all lowercase letters.
13. StrMove (1783) : Moves one `PChar` to another.
14. StrNew (1784) : Makes a copy of a `PChar` on the heap, and returns a pointer to this copy.
15. StrPos (1786) : Returns the position of one `PChar` string in another?
16. StrRScan (1786) : returns a pointer to the last occurrence of on `PChar` string in another one.
17. StrScan (1787) : returns a pointer to the first occurrence of on `PChar` string in another one.
18. StrUpper (1798) : Converts a `PChar` to all uppercase letters.

The subsequent functions are different from their counterparts in STRINGS, although the same examples can be used.

76.11 Date and time formatting characters.

Various date and time formatting routines accept a format string to format the date and or time. The following characters can be used to control the date and time formatting:

c Formats date using `shortdateformat` and formats time using `longtimeformat` if the time is not zero.

f Same as c, but adds the time even if it is zero.

d day of month

dd day of month (leading zero)

ddd day of week (abbreviation)

dddd day of week (full)

ddddd shortdateformat

dddddd longdateformat

m month or minutes if preceded by h or hh specifiers.

mm month or minutes if preceded by h or hh specifiers, with leading zero.

mmm month (abbreviation)

mmmm month (full)

y year (2 digits)

yy year (two digits)

yyyy year (with century)

h hour

hh hour (leading zero)

n minute

nn minute (leading zero)

s second

ss second (leading zero)

t shorttimeformat

tt longtimeformat

am/pm use 12 hour clock and display am and pm accordingly

a/p use 12 hour clock and display a and p accordingly

/ insert date separator

: insert time separator

"xx" literal text

'xx' literal text

z milliseconds

zzz milliseconds(leading zero)

[h] hours including the hours of the full days (i.e. can be > 24).

[hh] hours with leading zero, including the hours of the full days (i.e. can be > 24)

[n] minutes including the minutes of the full hours and days

[nn] minutes with leading zero, including the minutes of the full hours and days

[s] seconds including the seconds of the full minutes, hours and days.

[ss] seconds with leading zero, including the seconds of the full minutes, hours and days.

The forms in square brackets are only allowed if the fdoInterval (1657) option is included in the `Options` argument of FormatDateTime.

The date and time separators are taken from the DefaultFormatSettings (1664) record, unless a TFormatSettings (1657) record is passed to the FormatDateTime (1743) function.

Note that to include any of the above characters literally in the result string, they must be enclosed in double quotes.

See also: DefaultFormatSettings (1664), TFormatSettings (1657), FormatDateTime (1743), TFormatDateTimeOption (1657)

76.12 Formatting strings.

Functions for formatting strings.

Table 76.8:

Name	Description
AdjustLineBreaks (1669)	Convert line breaks to line breaks for system
FormatBuf (1742)	Format a buffer
Format (1735)	Format arguments in string
FmtStr (1734)	Format buffer
QuotedStr (1765)	Quote a string
StrFmt (1777)	Format arguments in a string
StrLFmt (1781)	Format maximum L characters in a string
TrimLeft (1803)	Remove whitespace at the left of a string
TrimRight (1804)	Remove whitespace at the right of a string
Trim (1803)	Remove whitespace at both ends of a string

76.13 String functions.

Functions for handling strings.

Table 76.9:

Name	Description
AnsiCompareStr (1670)	Compare two strings
AnsiCompareText (1671)	Compare two strings, case insensitive
AnsiExtractQuotedStr (1672)	Removes quotes from string
AnsiLastChar (1673)	Get last character of string
AnsiLowerCase (1674)	Convert string to all-lowercase
AnsiQuotedStr (1675)	Quotes a string
AnsiStrComp (1676)	Compare strings case-sensitive
AnsiStrIComp (1676)	Compare strings case-insensitive
AnsiStrLComp (1678)	Compare L characters of strings case sensitive
AnsiStrLIComp (1679)	Compare L characters of strings case insensitive
AnsiStrLastChar (1677)	Get last character of string
AnsiStrLower (1680)	Convert string to all-lowercase
AnsiStrUpper (1681)	Convert string to all-uppercase
AnsiUpperCase (1682)	Convert string to all-uppercase
AppendStr (1683)	Append 2 strings
AssignStr (1684)	Assign value of strings on heap
CompareStr (1689)	Compare two strings case sensitive
CompareText (1690)	Compare two strings case insensitive
DisposeStr (1703)	Remove string from heap
IsValidIdent (1761)	Is string a valid pascal identifier
LastDelimiter (1762)	Last occurrence of character in a string
LeftStr (1762)	Get first N characters of a string
LoadStr (1763)	Load string from resources
LowerCase (1763)	Convert string to all-lowercase
NewStr (1764)	Allocate new string on heap
RightStr (1768)	Get last N characters of a string
StrAlloc (1772)	Allocate memory for string
StrBufSize (1772)	Reserve memory for a string
StrDispose (1775)	Remove string from heap
StrPas (1785)	Convert PChar to pascal string
StrPCopy (1785)	Copy pascal string
StrPLCopy (1785)	Copy N bytes of pascal string
UpperCase (1814)	Convert string to all-uppercase

76.14 Constants, types and variables

76.14.1 Constants

```
ConfigExtension : string = '.cfg'
```

`ConfigExtension` is the default extension used by the GetAppConfigFile (1746) call. It can be set to any valid extension for the current OS.

```
CPUEndian = TEndian.Little
```

`CPUEndian` describes whether the current CPU is little or big endian.

```
DateDelta = 693594
```

Days between 1/1/0001 and 12/31/1899.

```
DriveDelim = DriveSeparator
```

`DriveDelim` refers to the system unit's `DriveSeparator` constant, it is for Delphi compatibility only.

```
EmptyStr : string = ''
```

Empty String Constant.

```
EmptyWideStr : WideString = ''
```

Empty wide string.

```
faAnyFile = $000001FF
```

Use this attribute in the FindFirst (1727) call to find all matching files.

```
faArchive = $00000020
```

Attribute of a file, meaning the file has the archive bit set. Used in TSearchRec (1660) and FindFirst (1727)

```
faCompressed = $00000800platform
```

`faTemporary` can be returned by FindFirst (1727) or FindNext (1728) to indicate that a returned file is compressed (on file systems that support this).

```
faDirectory = $00000010
```

Attribute of a file, meaning the file is a directory. Used in TSearchRec (1660) and FindFirst (1727)

```
faEncrypted = $00004000platform
```

`faEncrypted` can be returned by FindFirst (1727) or FindNext (1728) to indicate that a returned file is encrypted (on file systems that support this).

```
faHidden = $00000002platform
```

Attribute of a file, meaning the file is read-only. Used in TSearchRec (1660) and FindFirst (1727)

```
faNormal = $00000080
```

`faNormal` can be used in FindFirst (1727) to indicate that normal files must be included in the result.

```
faReadOnly = $00000001
```

Attribute of a file, meaning the file is read-only. Used in TSearchRec (1660) and FindFirst (1727)

```
faSymLink = $00000400platform
```

faSymLink means the file (as returned e.g. by FindFirst (1727)/FindNext (1728)), is a symlink. It's ignored under Windows.

```
faSysFile = $00000004platform
```

Attribute of a file, meaning the file is a system file. Used in TSearchRec (1660) and FindFirst (1727)

```
faTemporary = $00000100platform
```

faTemporary can be returned by FindFirst (1727) or FindNext (1728) to indicate that a returned file is a temporary file.

```
faVirtual = $00010000platform
```

faVirtual can be returned by FindFirst (1727) or FindNext (1728) to indicate that a returned file is virtual (on file systems that support this).

```
faVolumeId = $00000008deprecated; platform
```

Attribute of a file, meaning the entry contains the volume ID. Used in TSearchRec (1660) and FindFirst (1727)

```
feInvalidHandle = THandle(- 1)
```

feInvalidHandle is the return value of FileOpen (1721) in case of an error.

```
fmOpenRead = $0000
```

fmOpenRead is used in the FileOpen (1721) call to open a file in read-only mode.

```
fmOpenReadWrite = $0002
```

fmOpenReadWrite is used in the FileOpen (1721) call to open a file in read-write mode.

```
fmOpenWrite = $0001
```

fmOpenWrite is used in the FileOpen (1721) call to open a file in write-only mode.

```
fmShareCompat = $0000
```

fmOpenShareCompat is used in the FileOpen (1721) call OR-ed together with one of fmOpenReadWrite (1645), fmOpenRead (1645) or fmOpenWrite (1645),to open a file in a sharing modus that is equivalent to sharing implemented in MS-DOS.

```
fmShareDenyNone = $0040
```

fmShareDenyNone is used in the FileOpen (1721) call OR-ed together with one of fmOpenReadWrite (1645), fmOpenRead (1645) or fmOpenWrite (1645), to open a file so other processes can read/write the file as well.

```
fmShareDenyRead = $0030
```

fmOpenShareRead is used in the FileOpen (1721) call OR-ed together with one of fmOpenReadWrite (1645), fmOpenRead (1645) or fmOpenWrite (1645), to open a file so other processes cannot read from it.

This constant only works on Windows, because other operating systems do not support this constants.

```
fmShareDenyWrite = $0020
```

fmOpenShareDenyWrite is used in the FileOpen (1721) call OR-ed together with one of fmOpenReadWrite (1645), fmOpenRead (1645) or fmOpenWrite (1645), to open a file so other processes cannot write to it, they can only read.

```
fmShareExclusive = $0010
```

fmOpenShareExclusive is used in the FileOpen (1721) call OR-ed together with one of fmOpenReadWrite (1645), fmOpenRead (1645) or fmOpenWrite (1645), to open a file exclusively.

```
fsFromBeginning = 0
```

fsFromBeginning is used to indicate in the FileSeek (1723) call that a seek operation should be started at the start of the file.

```
fsFromCurrent = 1
```

fsFromBeginning is used to indicate in the FileSeek (1723) call that a seek operation should be started at the current position in the file.

```
fsFromEnd = 2
```

fsFromBeginning is used to indicate in the FileSeek (1723) call that a seek operation should be started at the last position in the file.

```
GUID_NULL : TGuid = '{00000000-0000-0000-0000-000000000000}'
```

NULL GUID constant.

```
HexDisplayPrefix : string = '$'
```

HexDisplayPrefix is used by the formatting routines to indicate that the number which follows the prefix is in Hexadecimal notation.

```
HoursPerDay = 24
```

Number of hours in a day.

```
INVALID_HANDLE_VALUE = DWORD(- 1)
```

```
JulianEpoch = TDateTime(- 2415018.5)
```

Starting point of the Julian calendar.

```
LeadBytes : Set of Char = []
```

`LeadBytes` contains the set of bytes that serve as lead byte in a MBCS string.

```
MaxCurrency : Currency = 922337203685477.5807
```

Maximum currency value.

```
MaxDateTime : TDateTime = 2958465.99999999
```

Maximum `TDateTime` value.

```
MAX_PATH = MaxPathLen
```

`MAX_PATH` is the maximum number of characters that a filename (including path) can contain on the current operating system.

```
MinCurrency : Currency = - 922337203685477.5808
```

Minimum Currency value.

```
MinDateTime : TDateTime = - 693593.99999999
```

Minimum `TDateTime` value.

```
MinsPerDay = HoursPerDay * MinsPerHour
```

Number of minutes per day.

```
MinsPerHour = 60
```

Number of minutes per hour.

```
MonthDays : Array[Boolean] of TDayTable = ((31, 28, 31, 30, 31, 30
  , 31, 31, 30, 31, 30, 31), (31, 29, 31, 30, 31, 30, 31, 31, 30, 31
  , 30, 31))
```

Array with number of days in the months for leap and non-leap years.

```
MSecsPerDay = SecsPerDay * MSecsPerSec
```

Number of milliseconds per day.

```
MSecsPerSec = 1000
```

Number of milliseconds per second.

```
NullStr : PString = @ EmptyStr
```

Pointer to an empty string.

```
PathDelim = DirectorySeparator
```

`PathDelim` refers to the system unit's `DirectorySeparator` constant, it is for Delphi compatibility only.

```
PathSep = PathSeparator
```

`PathSep` refers to the system unit's `PathSeparator` constant, it is for Delphi compatibility only.

```
pfBCB4Produced = $08000000
```

Not used in Free Pascal.

```
pfDelphi4Produced = $0C000000
```

Not used in Free Pascal.

```
pfDesignOnly = $00000002
```

Package is a design-time only package.

```
pfExeModule = $00000000
```

Package is an executable.

```
pfIgnoreDupUnits = $00000008
```

Ignore duplicate units in package.

```
pfLibraryModule = $80000000
```

Package is a library.

```
pfModuleTypeMask = $C0000000
```

Mask for module type flags.

```
pfNeverBuild = $00000001
```

Never-build flag was specified when compiling package.

```
pfPackageModule = $40000000
```

Package is a real package (not exe).

```
pfProducerMask = $0C000000
```

Mask for producer flags.

```
pfProducerUndefined = $04000000
```

Not used in Free Pascal.

```
pfRunOnly = $00000004
```

Package is a run-time only package.

```
pfV3Produced = $00000000
```

Not used in Free Pascal.

```
RTL_SIGBUS = 4
```

Bus error signal number (Unix only).

```
RTL_SIGDEFAULT = - 1
```

Default signal handler (Unix only).

```
RTL_SIGFPE = 1
```

Floating Point Error signal number (Unix only).

```
RTL_SIGILL = 3
```

Illegal instruction signal number (Unix only).

```
RTL_SIGINT = 0
```

INTERRUPT signal number (Unix only).

```
RTL_SIGLAST = RTL_SIGQUIT
```

Last signal number (Unix only).

```
RTL_SIGQUIT = 5
```

QUIT signal number (Unix only).

```
RTL_SIGSEGV = 2
```

Segmentation fault signal number (Unix only).

```
SecsPerDay = MinsPerDay * SecsPerMin
```

Number of seconds per day.

```
SecsPerHour = SecsPerMin * MinsPerHour
```

`SecsPerHour` is a constant indicating the number of seconds in an hour (3600).

```
SecsPerMin = 60
```

Number of seconds per minute.

```
SwitchChars = ['-']
```

The characters in this set will be used by the FindCmdLineSwitch (1726) function to determine whether a command-line argument is a switch (an option) or a value. If the first character of an argument is in `SwitchChars`, it will be considered an option or switch.

```
SysConfigDir : string = ''
```

`SysConfigDir` is the default system configuration directory. It is set at application startup by the sysutils initialization routines.

This directory may be returned by the GetAppConfigDir (1746) call on some systems.

```
ufImplicitUnit = $10
```

Unit was implicitly imported into package (did not appear in package contains list).

```
ufMainUnit = $01
```

Unit is the main unit of the package.

```
ufOrgWeakUnit = $08
```

Unit is the original weak packaged unit.

```
ufPackageUnit = $02
```

Unit is a packaged unit (appeared in package contains list).

```
ufWeakPackageUnit = ufPackageUnit or ufWeakUnit
```

Weak (original or not) packaged unit.

```
ufWeakUnit = $04
```

Unit is a weak packaged unit.

```
UnixDateDelta = Trunc(UnixEpoch)
```

Number of days between 1.1.1900 and 1.1.1970.

```
UnixEpoch = JulianEpoch + TDateTime(2440587.5)
```

Starting point of the unix calendar (1/1/1970).

76.14.2 Types

```
EHeapException = EHeapMemoryError
```

`EHeapMemoryError` is raised when an error occurs in the heap management routines.

```
ExceptClass = Class of Exception
```

`ExceptClass` is a Exception (1839) class reference.

```
Int128Rec = packed record
case Integer of
0: (
public
  Lo : QWord
  ;
  Hi : QWord;
);
1: (
public
  DWords : Array[0..3] of DWord;
)
  ;
2: (
public
  Words : Array[0..7] of Word;
);
3: (
public
  Bytes
  : Array[0..15] of Byte;
);
end
```

`Int128Rec` is a record defining a 128-bit integer. It is made up of 2 QWords or 4 DWords or 8 words or 16 bytes.

```
Int64Rec = packed record
case Integer of
0: (
public
  Lo : Cardinal
  ;
  Hi : Cardinal;
);
1: (
public
  Words : Array[0..3] of Word;
  );
2: (
public
  Bytes : Array[0..7] of Byte;
);
end
```

Int64Rec can be used to extract the parts of a Int64: the high and low cardinal, or a zero-based array of 4 words, or a zero based array of 8 bytes. Note that the meaning of the High and Low parts are different on various CPUs.

```
LongRec = packed record
case Integer of
0: (
public
  Lo : Word;
  Hi : Word;
);
1: (
public
  Bytes : Array[0..3] of Byte;
);
end
```

LongRec can be used to extract the parts of an long Integer: the high and low word, or the 4 separate bytes as a zero-based array of bytes. Note that the meaning (or ordering) of High and Low parts are different on various CPUs, and may differ from what is shown in the declaration: the ordering depends on the endianness of the CPU.

```
OWordRec = packed record
case Integer of
0: (
public
  Lo : QWord
  ;
  Hi : QWord;
);
1: (
public
  DWords : Array[0..3] of DWord;
)
  ;
2: (
public
  Words : Array[0..7] of Word;
);
3: (
public
  Bytes
  : Array[0..15] of Byte;
);
end
```

OWordRec is a record defining a 128-bit integer. It is made up of 2 QWords or 4 DWords or 8 words or 16 bytes.

```
PByteArray = ^TByteArray
```

Generic pointer to TByteArray (1653). Use to access memory regions as a byte array.

```
PDayTable = ^TDayTable
```

Pointer to TDayTable type.

```
PString = ObjPas.PString
```

Pointer to a ansistring.

```
PSysCharSet = ^TSysCharSet
```

Pointer to TSysCharSet (1661) type.

```
PWordarray = ^TWordArray
```

Generic pointer to TWordArray (1663). Use to access memory regions as a word array.

```
TArchitectures = Set of TOSVersion.TArchitecture
```

```
TArray<T> = Array of T
```

`TArray` is a generic array. It can be used to define dynamic arrays in generic functions or classes.

This definition is provided for Delphi compatibility only, it is not needed in Free Pascal, where 2 array types are equal if their element types are equal.

```
TBeepHandler = procedure
```

`TBeepHandler` is the prototype used by the OnBeep (1666) handler. This in turn is called by the Beep (1685) call to actually implement the beep functionality. The call takes no arguments.

```
TByteArray = Array[0..32767] of Byte
```

`TByteArray` is a generic array definition, mostly for use as a base type of the PByteArray (1652) type.

```
TByteBitIndex = 0..7
```

`TByteBitIndex` is the range of allowed bit indexes for a 8-bit unsigned integer type, it is used in the 8-bit helper TByteHelper (1855).

```
TBytes = Array of Byte
```

`TBytes` defines a dynamic array of bytes. This can be used to typecast e.g. strings to manipulate them byte for byte.

```
TCardinalBitIndex = 0..31
```

`TIntegerBitIndex` is the range of allowed bit indexes for a 32-bit unsigned integer type, it is used in the 32-bit helper TCardinalHelper (1859).

```
TCharArray = Array of Char
```

TCharArray is a dynamic array of characters.

```
TCompareOption = system.TCompareOption
```

TCompareOption is an alias for System.TCompareOption (1412).

```
TCompareOptions = system.TCompareOptions
```

TCompareOptions is an alias for System.TCompareOptions (1413).

```
TCreateGUIDFunc = function(out GUID: TGuid) : Integer
```

TCreateGUIDFunc is the prototype for a GUID creation handler. On return, the GUID argument should contain a new (unique) GUID. The return value of the function should be zero for success, nonzero for failure.

```
TDayTable = Array[1..12] of Word
```

Array of day names.

```
TEndian = ObjPas.TEndian
```

TEndian indicates whether the platform is little endian (LSB first), or big endian (MSB first).

```
TEventType = (etCustom,etInfo,etWarning,etError,etDebug)
```

Table 76.10: Enumeration values for type TEventType

Value	Explanation
etCustom	Custom log event, with application-specific meaning.
etDebug	Debug message.
etError	Error condition message.
etInfo	General information event message.
etWarning	Warning message.

TEventType is a type to be used by logging mechanisms (in particular, the TCustomApplication and TEventLog classes. It can be used to filter events, and write only certain types of event to the event log.

```
TEventTypes = Set of TEventType
```

TEventTypes is a set type of TEventType, defined for convenience. It is used in the custom application classes for logging purposes.

```
TExecuteFlags= Set of (ExecInheritsHandles)
```

Table 76.11: Enumeration values for type

Value	Explanation
ExecInheritsHandles	The new process inherits all (file) handles owned by the current process.

TExecuteFlags is a set of flags to influence the behaviour of the ExecuteProcess (1707) call.

```
TFileName = string
```

TFileName is used in the TSearchRec (1660) definition.

```
TFilenameCaseMatch = (mkNone,mkExactMatch,mkSingleMatch,mkAmbiguous
  )
```

Table 76.12: Enumeration values for type TFilenameCaseMatch

Value	Explanation
mkAmbiguous	More than one file will match the filename in a case-insensitive way.
mkExactMatch	The filename can be used to refer to a file on the system (findfirst will find it).
mkNone	No file was found.
mkSingleMatch	Exactly one match was found, but case didn't match.

TFilenameCaseMatch describes how ExpandFileNameCase (1709) found the file.

```
TFilenameCaseMatch = (mkNone,mkExactMatch,mkSingleMatch,mkAmbiguous
  )
```

Table 76.13: Enumeration values for type TFilenameCaseMatch

Value	Explanation
mkAmbiguous	More than one file will match the filename in a case-insensitive way.
mkExactMatch	The filename can be used to refer to a file on the system (findfirst will find it).
mkNone	No file was found.
mkSingleMatch	Exactly one match was found, but case didn't match.

TFilenameCaseMatch describes how ExpandFileNameCase (1709) found the file.

```
TFileRec = FileRec
```

Alias for FileRec (1635) for Delphi compatibility.

```
TFileSearchOption = (sfoImplicitCurrentDir,sfoStripQuotes)
```

Table 76.14: Enumeration values for type TFileSearchOption

Value	Explanation
sfoImplicitCurrentDir	Always search the current directory first, even if it is not specified.
sfoStripQuotes	Strip quotes from the components in the search path.

TFileSearchOption enumerates the options that can be used in the FileSearch call to control the behaviour of the search mechanism

```
TFileSearchOptions = Set of TFileSearchOption
```

TFileSearchOptions is a set of TFileSearchOption (1655) values, used in the FileSearch (1722) call when searching for files.

```
TFloatFormat = (ffGeneral,ffExponent,ffFixed,ffNumber,ffCurrency)
```

Table 76.15: Enumeration values for type TFloatFormat

Value	Explanation
ffCurrency	Monetary format.
ffExponent	Scientific format.
ffFixed	Fixed point format.
ffGeneral	General number format.
ffNumber	Fixed point format with thousand separator.

TFloatFormat is used to determine how a float value should be formatted in the FloatToText (1732) function.

```
TFloatRec = record
public
  Exponent : Integer;
  Negative : Boolean
  ;
  Digits : Array[0..18] of Char;
end
```

TFloatRec is used to describe a floating point value by the FloatToDecimal (1729) function.

```
TFloatValue = (fvExtended,fvCurrency,fvSingle,fvReal,fvDouble,fvComp
  )
```

Table 76.16: Enumeration values for type TFloatValue

Value	Explanation
fvComp	Comp value.
fvCurrency	Currency value.
fvDouble	Double value.
fvExtended	Extended value.
fvReal	Real value.
fvSingle	Single value.

TFloatValue determines which kind of value should be returned in the (untyped) buffer used by the TextToFloat (1800) function.

```
TFormatDateTimeOption = (fdoInterval)
```

Table 76.17: Enumeration values for type TFormatDateTimeOption

Value	Explanation
fdoInterval	Format the time as an interval, 24+hours are presented as such.

`TFormatDateTimeOption` enumerates possible options to the FormatDateTime (1743) routine.

```
TFormatDateTimeOptions = Set of TFormatDateTimeOption
```

`TFormatDateTimeOptions` is a set of TFormatDateTimeOptions (1657), and is used in the last argument of FormatDateTime (1743)

```
TFormatSettings = record
public
  CurrencyFormat : Byte;
  NegCurrFormat
  : Byte;
  ThousandSeparator : Char;
  DecimalSeparator : Char;
  CurrencyDecimals : Byte;
  DateSeparator : Char;
  TimeSeparator
  : Char;
  ListSeparator : Char;
  CurrencyString : string;
  ShortDateFormat
  : string;
  LongDateFormat : string;
  TimeAMString : string;
  TimePMString
  : string;
  ShortTimeFormat : string;
  LongTimeFormat : string;
  ShortMonthNames : TMonthNameArray;
  LongMonthNames : TMonthNameArray
  ;
  ShortDayNames : TWeekNameArray;
  LongDayNames : TWeekNameArray
  ;
  TwoDigitYearCenturyWindow : Word;
end
```

`TFormatSettings` is a record that contains a copy of all variables which determine formatting in the various string formatting routines. It is used to pass local copies of these values to the various formatting routines in a thread-safe way.

```
TGetAppNameEvent = function : string
```

This callback type is used by the OnGetApplicationName (1666) to return an alternative application name.

```
TGetTempDirEvent = function(Global: Boolean) : string
```

Function prototype for OnGetTempDir (1666) handler.

```
TGetTempFileEvent = function(const Dir: string; const Prefix: string
  )
                                : string
```

Function prototype for OnGetTempFile (1666) handler.

```
TGetVendorNameEvent = function : string
```

`TGetVendorNameEvent` is the function prototype for the OnGetVendorName (1666) callback, used by the VendorName (1815) function.

```
THandle = System.THandle
```

`THandle` refers to the definition of `THandle` in the system unit, and is provided for backward compatibility only.

```
TInt64BitIndex = 0..63
```

`TInt64BitIndex` is the range of allowed bit indexes for a 64-bit integer type, it is used in the 64-bit helper TInt64Helper (1884).

```
TIntegerBitIndex = 0..31
```

`TIntegerBitIndex` is the range of allowed bit indexes for a 32-bit signed integer type, it is used in the 32-bit helper TIntegerHelper (1888).

```
TIntegerSet = Set of
```

`TIntegerSet` is a generic integer subrange set definition whose size fits in a single integer.

```
TLocaleOptions = (loInvariantLocale,loUserLocale)
```

Table 76.18: Enumeration values for type TLocaleOptions

Value	Explanation
loInvariantLocale	Use system locale.
loUserLocale	Use user locale.

`TLocaleOptions` is used to indicate what locale should be used for some string operations such as UpperCase (1814), LowerCase (1763), CompareText (1690), SameText (1769). The following values are possible:

loInvariantLocale Use system locale.

loUserLocale Use user locale.

For consistent behaviour independent of the user or service applications use `loInvariantLocale`.

```
TLongIntBitIndex = TIntegerBitIndex
```

TLongIntBitIndex is the range of allowed bit indexes for a 32-bit signed integer type, it is used in the 32-bit helper TLongintHelper (1635).

```
TMbcsByteType = (mbSingleByte,mbLeadByte,mbTrailByte)
```

Table 76.19: Enumeration values for type TMbcsByteType

Value	Explanation
mbLeadByte	Uses lead-byte.
mbSingleByte	Single bytes.
mbTrailByte	Uses trailing byte.

Type of multi-byte character set.

```
TMonthNameArray = Array[1..12] of string
```

TMonthNameArray is used in the month long and short name arrays.

```
TNativeIntBitIndex = TInt64BitIndex
```

```
TNativeUIntBitIndex = TQwordBitIndex
```

```
TPlatforms = Set of TOSVersion.TPlatform
```

```
TProcedure = procedure
```

TProcedure is a general definition of a procedural callback.

```
TQwordBitIndex = 0..63
```

TQwordBitIndex is the range of allowed bit indexes for a 64-bit unsigned integer type, it is used in the 64-bit helper TQWordHelper (1906).

```
TReplaceFlags= Set of (rfReplaceAll,rfIgnoreCase)
```

Table 76.20: Enumeration values for type

Value	Explanation
rfIgnoreCase	Search case insensitive.
rfReplaceAll	Replace all occurrences of the search string with the replacement string.

TReplaceFlags determines the behaviour of the StringReplace (1778) function.

```
TSearchRec = TRawbyteSearchRec
```

TSearchRec is a search handle description record. It is initialized by a call to FindFirst (1727) and can be used to do subsequent calls to FindNext (1728). It contains the result of these function calls. It must be used to close the search sequence with a call to FindClose (1726).

Remark Not all fields of this record should be used. Some of the fields are for internal use only. (PathOnly for example, is only provided for Kylix compatibility)

```
TShortIntBitIndex = 0..7
```

TShortIntBitIndex is the range of allowed bit indexes for a 8-bit signed integer type, it is used in the 8-bit helper TShortIntHelper (1910).

```
TSignalState = (ssNotHooked,ssHooked,ssOverridden)
```

Table 76.21: Enumeration values for type TSignalState

Value	Explanation
ssHooked	A signal handler is set by the RTL code for the signal.
ssNotHooked	No signal handler is set for the signal.
ssOverridden	A signal handler was set for the signal by third-party code.

TSignalState indicates the state of a signal handler in a unix system for a particular signal.

```
TSmallIntBitIndex = 0..15
```

TSmallIntBitIndex is the range of allowed bit indexes for a 16-bit signed integer type, it is used in the 16-bit helper TSmallIntHelper (1922).

```
TStringArray = Array of string
```

TStringArray is a dynamic array of strings.

```
TStringBuilder = TANSISTRINGBUILDER
```

```
TStringSplitOptions = (None,ExcludeEmpty,ExcludeLastEmpty)
```

Table 76.22: Enumeration values for type TStringSplitOptions

Value	Explanation
ExcludeEmpty	Do not include empty strings in the result.
ExcludeLastEmpty	Only exclude the last element if it is empty.
None	No options specified. The default.

TStringSplitOptions describes the possible options for TStringHelper.Split (1943).

```
TSymLinkRec = TRawbyteSymLinkRec
```

TSymLinkRec contains information about the target of a symlink. Its actual type depends on the OS file API: it is either TRawbyteSymLinkRec (1826) or TUnicodeSymLinkRec (1828). Both types are used in the FileGetSymLinkTarget (1720) call.

TargetName #ShortDescr:

Size #ShortDescr:

Attr #ShortDescr:

Mode #ShortDescr:

TimeStamp #ShortDescr:

```
TSysCharSet = Set of AnsiChar
```

Generic set of characters type.

```
TSysLocale = record
public
  DefaultLCID : Integer;
  PriLangID :
  Integer;
  SubLangID : Integer;
case Byte of
1: (
public
  FarEast
  : Boolean;
  MiddleEast : Boolean;
);
2: (
public
  MBCS : Boolean
  ;
  RightToLeft : Boolean;
);
end
```

TSysLocale describes the current locale. If Fareast or MBCS is True, then the current locale uses a Multi-Byte Character Set. If MiddleEast or RightToLeft is True then words and sentences are read from right to left.

```
TSystemTime = record
public
  Year : Word;
  Month : Word;
  Day
  : Word;
  DayOfWeek : Word;
  Hour : Word;
  Minute : Word;
```

```
  Second
  : Word;
  MilliSecond : Word;
end
```

The System time structure contains the date/time in a human-understandable format.

```
TTerminateProc = function : Boolean
```

`TTerminateProc` is the procedural type which should be used when adding exit procedures.

```
TTextRec = TextRec
```

Alias for TextRec (1635) for Delphi compatibility.

```
TTimeStamp = record
public
  Time : LongInt;
  Date : LongInt;
end
```

`TTimeStamp` contains a timestamp, with the date and time parts specified as separate `TDateTime` values.

```
TUnicodeCharArray = Array of UnicodeChar
```

`TUnicodeCharArray` is a definition of an array of `UnicodeChar` elements.

```
TUseBoolStrs = (False,True)
```

Table 76.23: Enumeration values for type TUseBoolStrs

Value	Explanation
False	Numerical values should be used (0 and 1).
True	String representations should be used.

`TUseBoolStrs` is an enumerated type which indicates whether bool strs must be used when converting boolean to strings.

Because the identifiers `True` and `False` are already used for the system boolean type, You must use the scoped identifiers to get the `TUseBoolStrs` versions of these 2 identifiers. Failure to do so will result in an error.

```
TUTF8StringDynArray = Array of UTF8String
```

```
TWeekNameArray = Array[1..7] of string
```

`TWeekNameArray` is used in the day long and short name arrays.

```
TWordArray = Array[0..16383] of Word
```

TWordArray is a generic array definition, mostly for use as a base type of the PWordArray (1653) type.

```
TWordBitIndex = 0..15
```

TIntegerBitIndex is the range of allowed bit indexes for a 16-bit unsigned integer type, it is used in the 16-bit helper TWordHelper (1960).

```
WordRec = packed record
public
  Lo : Byte;
  Hi : Byte;
end
```

LongRec can be used to extract the parts of a word: the high and low byte. Note that the meaning of the High and Low parts are different on various CPUs.

76.14.3 Variables

```
CurrencyDecimals : Bytedeprecated
```

CurrencyDecimals is the number of decimals to be used when formatting a currency. It is used by the float formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use DefaultFormatSettings.CurrencyFormat (1664) instead.

```
CurrencyFormat : Bytedeprecated
```

CurrencyFormat is the default format string for positive currencies. It is used by the float formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use DefaultFormatSettings.CurrencyFormat (1664) instead.

```
CurrencyString : stringdeprecated
```

CurrencyString is the currency symbol for the current locale. It is used by the float formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use DefaultFormatSettings.CurrencyString (1664) instead.

```
DateSeparator : Chardeprecated
```

DateSeparator is the character used by various date/time conversion routines as the character that separates the day from the month and the month from the year in a date notation. It is used by the date formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use DefaultFormatSettings.DateSeparator (1664) instead.

```
DecimalSeparator : Chardeprecated
```

DecimalSeparator is used to display the decimal symbol in floating point numbers or currencies. It is used by the float formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use DefaultFormatSettings.DecimalSeparator (1664) instead.

```
DefaultFormatSettings : TFormatSettings = (CurrencyFormat: 1; NegCurrFormat
  : 5; ThousandSeparator: ','; DecimalSeparator: '.'; CurrencyDecimals
  : 2; DateSeparator: '-'; TimeSeparator: ':'; ListSeparator: ','; CurrencyString
  : '$'; ShortDateFormat: 'd/m/y'; LongDateFormat: 'dd" "mmmm" "yyyy'
  ; TimeAMString: 'AM'; TimePMString: 'PM'; ShortTimeFormat: 'hh:nn'
  ; LongTimeFormat: 'hh:nn:ss'; ShortMonthNames: ('Jan', 'Feb', 'Mar'
  , 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec');
  LongMonthNames: ('January', 'February', 'March', 'April', 'May',
  'June', 'July', 'August', 'September', 'October', 'November', 'December'
  ); ShortDayNames: ('Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat'
  ); LongDayNames: ('Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday'
  , 'Friday', 'Saturday'); TwoDigitYearCenturyWindow: 50)
```

DefaultFormatSettings contains the default settings for all type of formatting constants. If no thread-specific values are specified when a formatting function is called, this record is used as a default.

All other formatting constants refer to the fields of this variable using absolute addressing.

```
FalseBoolStrs : Array of string
```

FalseBoolStrs contains the strings that will result in a False return value by StrToBool (1787).

```
FormatSettings : TFormatSettings
```

FormatSettings is provided for Delphi compatibility, and refers to the DefaultFormatSettings (1664) variable.

```
ListSeparator : Chardeprecated
```

ListSeparator is the character used in lists of values. It is locale dependent.

```
LongDateFormat : stringdeprecated
```

LongDateFormat contains a template to format a date in a long format. It is used by the date formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use DefaultFormatSettings.LongDateFormat (1664) instead.

```
LongDayNames : TWeekNameArraydeprecated
```

LongDayNames is an array with the full names of days. It is used by the date formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

The array is indexed by values as returned by the DayOfWeek function.

This constant is deprecated. You should use DefaultFormatSettings.LongDayNames (1664) instead.

```
LongMonthNames : TMonthNameArraydeprecated
```

LongMonthNames is an array with the full names of months. It is used by the date formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use DefaultFormatSettings.LongMonthNames (1664) instead.

```
LongTimeFormat : stringdeprecated
```

LongTimeFormat contains a template to format a time in full notation. It is used by the time formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use DefaultFormatSettings.LongTimeFormat (1664) instead.

```
NegCurrFormat : Bytedeprecated
```

CurrencyFormat is the default format string for negative currencies. It is used by the float formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default:

0 Left parenthesis, currency symbol, amount, right parenthesis. Ex: ($1.2)

1 Negative sign, currency symbol, amount. Ex: -$1.2

2 Monetary symbol, negative sign, amount. Ex: $-1.2

3 Monetary symbol, amount, negative sign. Ex: $1.2-

4 Left parenthesis, amount, currency symbol, right parenthesis. Ex: (1.2$)

5 Negative sign, amount, currency symbol. Ex: -1.2$

5 6Amount, negative sign, currency symbol. Ex: 1.2-$

5 7Amount, currency symbol, negative sign. Ex: 1.2$-

5 8Negative sign, amount, space, currency symbol (as #5, adding a space before the currency symbol). Ex: -1.2 $

9 Negative sign, currency symbol, space, amount (as #1, adding a space after the currency symbol). Ex: -$ 1.2

10 Amount, space, currency symbol, negative sign (as #7, adding a space before the currency symbol). Ex: 1.2 $-

11 Monetary symbol, space, amount, negative sign (as #3, adding a space after the currency symbol). Ex: $ 1.2-

12 Monetary symbol, space, negative sign, amount (as #2, adding a space after the currency symbol). Ex: $ -1.2

13 Amount, negative sign, space, currency symbol (as #6, adding a space before the currency symbol). Ex: 1.2- $

14 Left parenthesis, currency symbol, space, amount, right parenthesis (as #0, adding a space after the currency symbol). Ex: ($ 1.2)

15 Left parenthesis, amount, space, currency symbol, right parenthesis (as ##4, adding a space before the currency symbol). Ex: (1.2 $)

```
OnBeep : TBeepHandler = Nil
```

OnBeep is calles whenever Beep is called. Beep contains no implementation to actually produce a beep, since there is no way to implement beep in a meaningful way for all possible implementations.

```
OnCreateGUID : TCreateGUIDFunc = Nil
```

OnCreateGUID can be set to point to a custom routine that creates GUID values. If set, the CreateGUID (1693) function will use it to obtain a GUID value. If it is not set, a default implementation using random values will be used to create the unique value. The function should return a valid GUID in the GUID parameter, and should return zero in case of success.

```
OnGetApplicationName : TGetAppNameEvent
```

By default, the configuration file routines GetAppConfigDir (1746) and GetAppConfigFile (1746) use a default application name to construct a directory or filename. This callback can be used to provide an alternative application name.

Since the result of this callback will be used to construct a filename, care should be taken that the returned name does not contain directory separator characters or characters that cannot appear in a filename.

```
OnGetTempDir : TGetTempDirEvent
```

OnGetTempDir can be used to provide custom behaviour for the GetTempDir (1752) function. Note that the returned name should have a trailing directory delimiter character.

```
OnGetTempFile : TGetTempFileEvent
```

OnGetTempDir can be used to provide custom behaviour for the GetTempFileName (1752) function. Note that the values for Prefix and Dir should be observed.

```
OnGetVendorName : TGetVendorNameEvent
```

OnGetVendorName must be set in order for VendorName (1815) to return a value. It will then be used in GetAppConfigDir (1746) and GetAppConfigFile (1746) to determine the configuration directory. Set it to a callback that returns the actual vendor name for the application.

```
OnShowException : procedure(Msg: ShortString)
```

OnShowException is the callback that ShowException (1770) uses to display a message in a GUI application. For GUI applications, this variable should always be set. Note that no memory may be available when this callback is called, so the callback should already have all resources it needs, when the callback is set.

```
ShortDateFormat : stringdeprecated
```

ShortDateFormat contains a template to format a date in a short format. It is used by the date formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use DefaultFormatSettings.ShortDateFormat (1664) instead.

```
ShortDayNames : TWeekNameArraydeprecated
```

ShortDayNames is an array with the abbreviated names of days. It is used by the date formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

The array is indexed by values as returned by the DayOfWeek function.

This constant is deprecated. You should use DefaultFormatSettings.ShortDayNames (1664) instead.

```
ShortMonthNames : TMonthNameArraydeprecated
```

ShortMonthNames is an array with the abbreviated names of months. It is used by the date formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use DefaultFormatSettings.ShortMonthNames (1664) instead.

```
ShortTimeFormat : stringdeprecated
```

ShortTimeFormat contains a template to format a time in a short notation. It is used by the time formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use DefaultFormatSettings.ShortTimeFormat (1664) instead.

```
SysLocale : TSysLocale
```

SysLocale is initialized by the initialization code of the SysUtils unit. For an explanation of the fields, see TSysLocale (1661)

```
ThousandSeparator : Chardeprecated
```

ThousandSeparator is used to separate groups of thousands in floating point numbers or currencies. It is used by the float formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use DefaultFormatSettings.ThousandSeparator (1664) instead.

```
TimeAMString : stringdeprecated
```

TimeAMString is used to display the AM symbol in the time formatting routines. It is used by the time formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use DefaultFormatSettings.TimeAMString (1664) instead.

```
TimePMString : stringdeprecated
```

TimePMString is used to display the PM symbol in the time formatting routines. It is used by the time formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use DefaultFormatSettings.TimePMString (1664) instead.

```
TimeSeparator : Chardeprecated
```

TimeSeparator is used by the time formatting routines to separate the hours from the minutes and the minutes from the seconds. It is used by the time formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

```
TrueBoolStrs : Array of string
```

TrueBoolStrs contains the strings that will result in a True return value by StrToBool (1787).

```
TwoDigitYearCenturyWindow : Word
```

Window to determine what century 2 digit years are in.

76.15 Procedures and functions

76.15.1 AbandonSignalHandler

Synopsis: Abandon the signal handler.

Declaration: `procedure AbandonSignalHandler(RtlSigNum: Integer)`

Visibility: default

Description: AbandonSignalHandler tells the system routines that they should not re-install the signal handler for signal RtlSigNum under any circumstances. Normally, signal handlers are re-set when they are called. If AbandonSignalHandler has been called for a signal that is handled by the system code, the signal will not be re-set again.

76.15.2 Abort

Synopsis: Abort program execution.

Declaration: `procedure Abort`

Visibility: default

Description: Abort raises an EAbort (1829) exception.

See also: EAbort (1829)

76.15.3 AddDisk

Synopsis: Add a disk to the list of known disks (Unix only).

Declaration: `function AddDisk(const path: string) : Byte`

Visibility: default

Description: On Unix-like platforms both the DiskFree (1701) and DiskSize (1702) functions need a file on the specified drive, since is required for the statfs system call.

These filenames are set in drivestr[0..26], and the first 4 have been preset to :

Disk 0'.' default drive - hence current directory is used.

Disk 1'`/fd0/.'` floppy drive 1.

Disk 2'`/fd1/.'` floppy drive 2.

Disk 3'`/'`C: equivalent of DOS is the root partition.

Drives 4..26 can be set by your own applications with the `AddDisk` call.

The `AddDisk` call adds `Path` to the names of drive files, and returns the number of the disk that corresponds to this drive. If you add more than 21 drives, the count is wrapped to 4.

Errors: None.

See also: DiskFree (1701), DiskSize (1702)

76.15.4 AddTerminateProc

Synopsis: Add a procedure to the exit chain.

Declaration:
```
procedure AddTerminateProc(TermProc: TTerminateProc)
```

Visibility: default

Description: `AddTerminateProc` adds `TermProc` to the list of exit procedures. When the program exits, the list of exit procedures is run over, and all procedures are called one by one, in the reverse order that they were added to the exit chain.

Errors: If no memory is available on the heap, an exception may be raised.

See also: TTerminateProc (1662), CallTerminateProcs (1687)

76.15.5 AdjustLineBreaks

Synopsis: Convert possible line-endings to the currently valid line ending.

Declaration:
```
function AdjustLineBreaks(const S: string) : string
function AdjustLineBreaks(const S: string; Style: TTextLineBreakStyle)
                          : string
```

Visibility: default

Description: `AdjustLineBreaks` will change all occurrences of `#13` and `#10` characters with the correct line-ending characters for the current platform, or the specified platform if the 'Style' parameter was specified.

The platform-specific line-ending characters are:

- #13#10 on Windows and DOS
- #13 on Classic Mac OS before OS X
- #10 on Unix clones / Linux / OS X / macOS.

Errors: None.

See also: AnsiCompareStr (1670), AnsiCompareText (1671)

Listing: ./examples/sysutex/ex48.pp

```
Program Example48;

{ This program demonstrates the AdjustLineBreaks function }

Uses sysutils;

Const
  S = 'This is a string'#13'with embedded'#10'linefeed and'+
      #13'CR characters';

Begin
  WriteLn (AdjustLineBreaks(S));
End.
```

76.15.6 AnsiCompareFileName

Synopsis: Compare 2 filenames.

Declaration: `function AnsiCompareFileName(const S1: string; const S2: string) : SizeInt`

Visibility: default

Description: `AnsiCompareFileName` compares 2 filenames `S1` and `S2`, and returns

< 0if `S1<S2`.

= 0if `S1=S2`.

> 0if `S1>S2`.

The function actually checks `FileNameCaseSensitive` and returns the result of AnsiCompareStr (1670) or AnsiCompareText (1671) depending on whether `FileNameCaseSensitive` is `True` or `False`

Errors: None.

See also: AnsiCompareStr (1670), AnsiCompareText (1671), AnsiLowerCaseFileName (1674)

76.15.7 AnsiCompareStr

Synopsis: Compare 2 ansistrings, case sensitive, ignoring accents characters.

Declaration: `function AnsiCompareStr(const S1: string; const S2: string) : Integer`

Visibility: default

Description: `AnsiCompareStr` compares two strings and returns the following result:

< 0if `S1<S2`.

0if `S1=S2`.

> 0if `S1>S2`.

The comparison takes into account Ansi characters, i.e. it takes care of strange accented characters. Contrary to AnsiCompareText (1671), the comparison is case sensitive.

Remark A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: AdjustLineBreaks (1669), AnsiCompareText (1671)

Listing: ./examples/sysutex/ex49.pp

```
Program Example49;

{ This program demonstrates the AnsiCompareStr function }
{$H+}

Uses sysutils;

Procedure TestIt (S1,S2 : String);

Var R : Longint;

begin
  R:=AnsiCompareStr(S1,S2);
  Write ('"',S1,'" is ');
  If R<0 then
    write ('less than ')
  else If R=0 then
    Write ('equal to ')
  else
    Write ('larger than ');
  WriteIn ('"',S2,'"');
end;

Begin
  Testit('One string','One smaller string');
  Testit('One string','one string');
  Testit('One string','One string');
  Testit('One string','One tall string');
End.
```

76.15.8 AnsiCompareText

Synopsis: Compare 2 ansistrings, case insensitive, ignoring accents characters.

Declaration: `function AnsiCompareText(const S1: string; const S2: string) : Integer`

Visibility: default

Description: `AnsiCompareText` compares two strings and returns the following result:

<0if `S1<S2`.

0if `S1=S2`.

>0if `S1>S2`.

the comparison takes into account Ansi characters, i.e. it takes care of strange accented characters. Contrary to AnsiCompareStr (1670), the comparison is case insensitive.

Remark A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: AdjustLineBreaks (1669), AnsiCompareText (1671)

Listing: ./examples/sysutex/ex50.pp

```
Program Example49;

{ This program demonstrates the AnsiCompareText function }
{$H+}

Uses sysutils;

Procedure TestIt (S1,S2 : String);

Var R : Longint;

begin
  R:=AnsiCompareText(S1,S2);
  Write ('"',S1,'" is ');
  If R<0 then
    write ('less than ')
  else If R=0 then
    Write ('equal to ')
  else
    Write ('larger than ');
  WriteIn ('"',S2,'"');
end;

Begin
  Testit('One string','One smaller string');
  Testit('One string','one string');
  Testit('One string','One string');
  Testit('One string','One tall string');
End.
```

76.15.9 AnsiDequotedStr

Synopsis: Extract string without quotes.

Declaration: `function AnsiDequotedStr(const S: string; AQuote: Char) : string`

Visibility: default

Description: `AnsiDequotedStr` is, similar to AnsiExtractQuotedStr (1672), a method to extract a string that was quoted using AnsiQuotedStr (1675). The string `S` must start and end with the quote character `AQuote` (they will be removed from the result) and all double occurrences of the quote character `AQuote` will be reduced to a single quote character.

Errors: If the string does not start and end with the quote character or has length less than 2, the original string is returned.

See also: AnsiExtractQuotedStr (1672), AnsiQuotedStr (1675)

76.15.10 AnsiExtractQuotedStr

Synopsis: Removes the first quoted string from a string.

Declaration: `function AnsiExtractQuotedStr(var Src: PChar; Quote: Char) : string`

Visibility: default

Description: `AnsiExtractQuotedStr` returns the first quoted string in `Src`, and deletes the result from `Src`. The resulting string has with `Quote` characters removed from the beginning and end of the string (if they are present), and double `Quote` characters replaced by a single `Quote` characters. As such, it reverses the action of AnsiQuotedStr (1675).

Errors: None.

See also: AnsiQuotedStr (1675)

Listing: ./examples/sysutex/ex51.pp

```
Program Example51;
{ This program demonstrates the AnsiQuotedStr function }
Uses sysutils;

Var
  S : AnsiString;
  P : PChar;

Begin
 S:='He said "Hello" and walked on';
 P:=Pchar(S);
 S:=AnsiQuotedStr(P,'"');
 Writeln (S);
 P:=Pchar(S);
 Writeln(AnsiExtractQuotedStr(P,'"'));
End.
```

76.15.11 AnsiLastChar

Synopsis: Return a pointer to the last character of a string.

Declaration: `function AnsiLastChar(const S: string) : PChar`

Visibility: default

Description: This function returns a pointer to the last character of `S`.

Remark A widestring manager must be installed in order for this function to work correctly with various character sets. If none is installed, this function is the same as `@S[Length[S]]`.

Errors: None.

See also: AnsiStrLastChar (1677)

Listing: ./examples/sysutex/ex52.pp

```
Program Example52;

{ This program demonstrates the AnsiLastChar function }

Uses sysutils;

Var S : AnsiString;
    L : Longint;

Begin
```

```
  S:='This is an ansistring.';
  WriteLn ('Last character of S is : ',AnsiLastChar(S));
  L:=Longint(AnsiLastChar(S))-Longint(@S[1])+1;
  WriteLn ('Length of S is : ',L);
End.
```

76.15.12 AnsiLowerCase

Synopsis: Return a lowercase version of a string.

Declaration: `function AnsiLowerCase(const s: string) : string`

Visibility: default

Description: `AnsiLowerCase` converts the string `S` to lowercase characters and returns the resulting string. It takes into account the operating system language settings when doing this, so special characters are converted correctly as well.

Remark A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: AnsiUpperCase (1682), AnsiStrLower (1680), AnsiStrUpper (1681)

Listing: ./examples/sysutex/ex53.pp

```
Program Example53;

{ This program demonstrates the AnsiLowerCase function }

Uses sysutils;

Procedure Testit (S : String);

begin
 WriteLn (S,' -> ',AnsiLowerCase(S))
end;

Begin
  Testit('AN UPPERCASE STRING');
  Testit('Some mixed STring');
  Testit('a lowercase string');
End.
```

76.15.13 AnsiLowerCaseFileName

Synopsis: Convert filename to lowercase.

Declaration: `function AnsiLowerCaseFileName(const s: string) : string`

Visibility: default

Description: `AnsiLowerCaseFileName` simply returns the result of

```
AnsiLowerCase(S);
```

See also: AnsiLowerCase (1674), AnsiCompareFileName (1670), AnsiUpperCaseFileName (1682)

76.15.14 AnsiPos

Synopsis: Return Position of one ansistring in another.

Declaration: `function AnsiPos(const substr: string; const s: string) : SizeInt`

Visibility: default

Description: `AnsiPos` does the same as the standard `Pos` function.

See also: AnsiStrPos (1680), AnsiStrScan (1681), AnsiStrRScan (1681)

76.15.15 AnsiQuotedStr

Synopsis: Return a quoted version of a string.

Declaration: `function AnsiQuotedStr(const S: string; Quote: Char) : string`

Visibility: default

Description: `AnsiQuotedString` quotes the string `S` and returns the result. This means that it puts the `Quote` character at both the beginning and end of the string and replaces any occurrence of `Quote` in `S` with 2 `Quote` characters. The action of `AnsiQuotedString` can be reversed by AnsiExtractQuotedStr (1672).

For an example, see AnsiExtractQuotedStr (1672)

Errors: None.

See also: AnsiExtractQuotedStr (1672)

76.15.16 AnsiSameStr

Synopsis: Checks whether 2 strings are the same (case sensitive).

Declaration: `function AnsiSameStr(const s1: string; const s2: string) : Boolean`

Visibility: default

Description: `SameText` calls AnsiCompareStr (1670) with `S1` and `S2` as parameters and returns `True` if the result of that call is zero, or `False` otherwise.

Remark A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: AnsiCompareStr (1670), SameText (1769), AnsiSameText (1675)

76.15.17 AnsiSameText

Synopsis: Checks whether 2 strings are the same (case insensitive).

Declaration: `function AnsiSameText(const s1: string; const s2: string) : Boolean`

Visibility: default

Description: `SameText` calls AnsiCompareText (1671) with `S1` and `S2` as parameters and returns `True` if the result of that call is zero, or `False` otherwise.

See also: AnsiCompareText (1671), SameText (1769), AnsiSameStr (1675)

76.15.18 AnsiStrComp

Synopsis: Compare two null-terminated strings. Case sensitive.

Declaration: `function AnsiStrComp(S1: PChar; S2: PChar) : Integer`

Visibility: default

Description: `AnsiStrComp` compares 2 `PChar` strings, and returns the following result:

<0if `S1<S2`.

0if `S1=S2`.

>0if `S1>S2`.

The comparison of the two strings is case-sensitive.

Remark A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: AnsiCompareText (1671), AnsiCompareStr (1670)

Listing: ./examples/sysutex/ex54.pp

```
Program Example54;

{ This program demonstrates the AnsiStrComp function }

Uses sysutils;

Procedure TestIt (S1,S2 : Pchar);

Var R : Longint;

begin
  R:=AnsiStrComp(S1,S2);
  Write ('"',S1,'" is ');
  If R<0 then
    write ('less than ')
  else If R=0 then
    Write ('equal to ')
  else
    Write ('larger than ');
  WriteIn ('"',S2,'"');
end;

Begin
  Testit('One string','One smaller string');
  Testit('One string','one string');
  Testit('One string','One string');
  Testit('One string','One tall string');
End.
```

76.15.19 AnsiStrIComp

Synopsis: Compare two null-terminated strings. Case insensitive.

Declaration: `function AnsiStrIComp(S1: PChar; S2: PChar) : Integer`

Visibility: default

Description: `AnsiStrIComp` compares 2 `PChar` strings, and returns the following result:

<0if `S1<S2`.

0if `S1=S2`.

>0if `S1>S2`.

The comparison of the two strings is case-insensitive.

Remark A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: AnsiCompareText (1671), AnsiCompareStr (1670)

Listing: ./examples/sysutex/ex55.pp

```
Program Example55;

{ This program demonstrates the AnsiStrIComp function }

Uses sysutils;

Procedure TestIt (S1,S2 : Pchar);

Var R : Longint;

begin
  R:=AnsiStrIComp(S1,S2);
  Write ('"',S1,'" is ');
  If R<0 then
    write ('less than ')
  else If R=0 then
    Write ('equal to ')
  else
    Write ('larger than ');
  WriteIn ('"',S2,'"');
end;

Begin
  Testit('One string','One smaller string');
  Testit('One string','one string');
  Testit('One string','One string');
  Testit('One string','One tall string');
End.
```

76.15.20 AnsiStrLastChar

Synopsis: Return a pointer to the last character of a string.

Declaration: `function AnsiStrLastChar(Str: PChar) : PChar`

Visibility: default

Description: Return a pointer to the last character of the null-terminated string.

Remark A widestring manager must be installed in order for this function to work correctly with various character sets. If none is installed, this function is the same as `@S[Length[S]]`.

Errors: None.

See also: AnsiCompareText (1671), AnsiCompareStr (1670)

Listing: ./examples/sysutex/ex56.pp

```
Program Example56;

{ This program demonstrates the AnsiStrLComp function }

Uses sysutils;

Procedure TestIt (S1,S2 : Pchar; L : longint);

Var R : Longint;

begin
  R:=AnsiStrLComp(S1,S2,L);
  Write ('First ',L,' characters of "',S1,'" are ');
  If R<0 then
    write ('less than ')
  else If R=0 then
    Write ('equal to ')
  else
    Write ('larger than ');
  WriteIn ('those of "',S2,'"');
end;

Begin
  Testit('One string','One smaller string',255);
  Testit('One string','One String',4);
  Testit('One string','1 string',0);
  Testit('One string','One string.',9);
End.
```

76.15.21 AnsiStrLComp

Synopsis: Compare a limited number of characters of 2 strings.

Declaration: `function AnsiStrLComp(S1: PChar; S2: PChar; MaxLen: SizeUInt) : Integer`

Visibility: default

Description: `AnsiStrLComp` functions the same as AnsiStrComp (1676), but compares at most `MaxLen` characters. If the first `MaxLen` characters in both strings are the same, then zero is returned.

Note that this function processes embedded null characters, treating them as a normal character.

Errors: None.

See also: AnsiStrComp (1676), AnsiStrIComp (1676), AnsiStrLIComp (1679)

76.15.22 AnsiStrLIComp

Synopsis: Compares a given number of characters of a string, case insensitive.

Declaration: `function AnsiStrLIComp(S1: PChar; S2: PChar; MaxLen: SizeUInt) : Integer`

Visibility: default

Description: `AnsiStrLIComp` compares the first `Maxlen` characters of 2 `PChar` strings, `S1` and `S2`, and returns the following result:

<0if `S1<S2`.

0if `S1=S2`.

>0if `S1>S2`.

The comparison of the two strings is case-insensitive.

Remark A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: AnsiCompareText (1671), AnsiCompareStr (1670)

Listing: ./examples/sysutex/ex57.pp

```
Program Example57;

{ This program demonstrates the AnsiStrLIComp function }

Uses sysutils;

Procedure TestIt (S1,S2 : Pchar; L : longint);

Var R : Longint;

begin
  R:=AnsiStrLIComp(S1,S2,L);
  Write ('First ',L,' characters of "',S1,'" are ');
  If R<0 then
    write ('less than ')
  else If R=0 then
    Write ('equal to ')
  else
    Write ('larger than ');
  WriteIn ('those of "',S2,'"');
end;

Begin
  Testit('One string','One smaller string',255);
  Testit('ONE STRING','one String',4);
  Testit('One string','1 STRING',0);
  Testit('One STRING','one string.',9);
End.
```

76.15.23 AnsiStrLower

Synopsis: Convert a null-terminated string to all-lowercase characters.

Declaration: `function AnsiStrLower(Str: PChar) : PChar`

Visibility: default

Description: `AnsiStrLower` converts the PChar `Str` to lowercase characters and returns the resulting pchar. Note that `Str` itself is modified, not a copy, as in the case of AnsiLowerCase (1674). It takes into account the operating system language settings when doing this, so special characters are converted correctly as well.

Remark A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: AnsiStrUpper (1681), AnsiLowerCase (1674)

Listing: ./examples/sysutex/ex59.pp

```
Program Example59;

{ This program demonstrates the AnsiStrLower function }

Uses sysutils,strings;

Procedure Testit (S : Pchar);

begin
 WriteIn (S,' -> ',AnsiStrLower(StrNew(S)))
end;

Begin
  Testit('AN UPPERCASE STRING');
  Testit('Some mixed STring');
  Testit('a lowercase string');
End.
```

76.15.24 AnsiStrPos

Synopsis: Return position of one null-terminated substring in another.

Declaration: `function AnsiStrPos(str: PChar; substr: PChar) : PChar`

Visibility: default

Description: `AnsiStrPos` returns a pointer to the first occurrence of `SubStr` in `Str`. If `SubStr` does not occur in `Str` then `Nil` is returned.

Errors: An access violation may occur if either `Str` or `SubStr` point to invalid memory.

See also: AnsiPos (1675), AnsiStrScan (1681), AnsiStrRScan (1681)

76.15.25 AnsiStrRScan

Synopsis: Find last occurrence of a character in a null-terminated string.

Declaration: `function AnsiStrRScan(Str: PChar; Chr: Char) : PChar`

Visibility: default

Description: `AnsiStrPos` returns a pointer to the *last* occurrence of the character `Chr` in `Str`. If `Chr` does not occur in `Str` then `Nil` is returned.

Errors: An access violation may occur if `Str` points to invalid memory.

See also: AnsiPos (1675), AnsiStrScan (1681), AnsiStrPos (1680)

76.15.26 AnsiStrScan

Synopsis: Find first occurrence of a character in a null-terminated string.

Declaration: `function AnsiStrScan(Str: PChar; Chr: Char) : PChar`

Visibility: default

Description: `AnsiStrPos` returns a pointer to the *first* occurrence of the character `Chr` in `Str`. If `Chr` does not occur in `Str` then `Nil` is returned.

Errors: An access violation may occur if `Str` points to invalid memory.

See also: AnsiPos (1675), AnsiStrScan (1681), AnsiStrPos (1680)

76.15.27 AnsiStrUpper

Synopsis: Convert a null-terminated string to all-uppercase characters.

Declaration: `function AnsiStrUpper(Str: PChar) : PChar`

Visibility: default

Description: `AnsiStrUpper` converts the `PCharStr` to uppercase characters and returns the resulting string. Note that `Str` itself is modified, not a copy, as in the case of AnsiUpperCase (1682). It takes into account the operating system language settings when doing this, so special characters are converted correctly as well.

Remark A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: AnsiUpperCase (1682), AnsiStrLower (1680), AnsiLowerCase (1674)

Listing: ./examples/sysutex/ex60.pp

```
Program Example60;

{ This program demonstrates the AnsiStrUpper function }

Uses sysutils, strings;

Procedure Testit (S : Pchar);
```

```
begin
 WriteLn (S,' -> ',AnsiStrUpper(StrNew(S)))
end;

Begin
  Testit('AN UPPERCASE STRING');
  Testit('Some mixed STring');
  Testit('a lowercase string');
End.
```

76.15.28 AnsiUpperCase

Synopsis: Return an uppercase version of a string, taking into account special characters.

Declaration: `function AnsiUpperCase(const s: string) : string`

Visibility: default

Description: `AnsiUpperCase` converts the string `S` to uppercase characters and returns the resulting string. It takes into account the operating system language settings when doing this, so special characters are converted correctly as well.

Remark A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: AnsiStrUpper (1681), AnsiStrLower (1680), AnsiLowerCase (1674)

Listing: ./examples/sysutex/ex61.pp

```
Program Example60;

{ This program demonstrates the AnsiUpperCase function }

Uses sysutils;

Procedure Testit (S : String);

begin
 WriteLn (S,' -> ',AnsiUpperCase(S))
end;

Begin
  Testit('AN UPPERCASE STRING');
  Testit('Some mixed STring');
  Testit('a lowercase string');
End.
```

76.15.29 AnsiUpperCaseFileName

Synopsis: Convert filename to uppercase.

Declaration: `function AnsiUpperCaseFileName(const s: string) : string`

Visibility: default

Description: `AnsiUpperCaseFileName` simply returns the result of

```
AnsiUpperCase(S);
```

See also: AnsiUpperCase (1682), AnsiCompareFileName (1670), AnsiLowerCaseFileName (1674)

76.15.30 AppendStr

Synopsis: Append one ansistring to another.

Declaration: `procedure AppendStr(var Dest: string; const S: string)`

Visibility: default

Description: `AppendStr` appends `S` to Dest.

This function is provided for Delphi compatibility only, since it is completely equivalent to `Dest:=Dest+S`.

Errors: None.

See also: AssignStr (1684), NewStr (1764), DisposeStr (1703)

Listing: ./examples/sysutex/ex62.pp

```
Program Example62;

{ This program demonstrates the AppendStr function }

Uses sysutils;

Var S : AnsiString;

Begin
  S:='This is an ';
  AppendStr(S,'AnsiString');
  Writeln ('S = "',S,'"');
End.
```

76.15.31 ApplicationName

Synopsis: Return a default application name.

Declaration: `function ApplicationName : string`

Visibility: default

Description: `ApplicationName` returns the name of the current application. Standard this is equal to the filename part minus extension of `ParamStr(0)`, but it can be customized by setting the OnGetApplicationName (1666) callback.

Note that the returned value is only the `name` portion. It does not contain any path or file extension.

Errors: None.

See also: GetAppConfigDir (1746), OnGetApplicationName (1666), GetAppConfigFile (1746), ConfigExtension (1643)

76.15.32 ArrayOfConstToStr

Declaration: `function ArrayOfConstToStr(Args: Array of const; aSeparator: Char; aQuoteBegin: Char; aQuoteEnd: Char) : UTF8String`

Visibility: default

76.15.33 ArrayOfConstToStrArray

Declaration: `function ArrayOfConstToStrArray(Args: Array of const) : TUTF8StringDynArray`

Visibility: default

76.15.34 AssignStr

Synopsis: Assigns an ansistring to a null-terminated string.

Declaration: `procedure AssignStr(var P: PString; const S: string)`

Visibility: default

Description: `AssignStr` allocates `S` to `P`. The old value of `P` is disposed of.

This function is provided for Delphi compatibility only. `AnsiStrings` are managed on the heap and should be preferred to the mechanism of dynamically allocated strings.

Errors: None.

See also: NewStr (1764), AppendStr (1683), DisposeStr (1703)

Listing: ./examples/sysutex/ex63.pp

```
Program Example63;

{ This program demonstrates the AssignStr function }
{$H+}

Uses sysutils;

Var P : PString;

Begin
 P:=NewStr('A first AnsiString');
 Writeln ('Before: P = "',P^,'"');
 AssignStr(P,'A Second ansistring');
 Writeln ('After : P = "',P^,'"');
 DisposeStr(P);
End.
```

76.15.35 BCDToInt

Synopsis: Convert a BCD coded integer to a normal integer.

Declaration: `function BCDToInt(Value: Integer) : Integer`

Visibility: default

Description: `BCDToInt` converts a `BCD` coded integer to a normal integer.

Errors: None.

See also: StrToInt (1793), IntToStr (1758)

Listing: ./examples/sysutex/ex64.pp

```
Program Example64;

{ This program demonstrates the BCDToInt function }

Uses sysutils;

Procedure Testit ( L : longint);
begin
  WriteLn (L,' -> ',BCDToInt(L));
end;

Begin
  Testit(10);
  Testit(100);
  TestIt(23);
End.
```

76.15.36 Beep

Synopsis: Sound the system bell.

Declaration: `procedure Beep`

Visibility: default

Description: `Beep` sounds the system bell, if one is available. The actual beep is produced by the OnBeep (1666) callback. The `Sysutils` unit itself contains no implementation of this call.

76.15.37 BoolToStr

Synopsis: Convert a boolean value to a string.

Declaration:
```
function BoolToStr(B: Boolean; UseBoolStrs: Boolean) : string
function BoolToStr(B: Boolean; const TrueS: string;
                   const FalseS: string) : string
```

Visibility: default

Description: `BoolToStr` converts the boolean `B` to a string representation. Which string this is depends on the way the function is called: You can provide the strings to use for the `True` or `False` in the parameters `TrueS` and `FalseS`, or you can tell the function to use the first string in the TrueBoolStrs (1668) and FalseBoolStrs (1664) arrays by setting the `UseBoolStrs` parameter to `True`

The default (Delphi compatible) behaviour is to use `-1` for `True` and `0` for `False`.

Errors: None.

See also: StrToBool (1787), TrueBoolStrs (1668), FalseBoolStrs (1664)

76.15.38 ByteLength

Synopsis: Length (in bytes) of a unicodestring.

Declaration:
```
function ByteLength(const S: UnicodeString) : Integer
```

Visibility: default

Description: `ByteLength` returns the length of a unicodestring in bytes. This equals the character length of the string (Length (1635)) multiplied by the number of bytes per character (2).

See also: Length (1635)

76.15.39 BytesOf

Synopsis: Return the bytes of a string in a `TBytes` array.

Declaration:
```
function BytesOf(const Val: RawByteString) : TBytes
function BytesOf(const Val: AnsiChar) : TBytes
function BytesOf(const Val: UnicodeString) : TBytes;  Overload
function BytesOf(const Val: WideChar) : TBytes;  Overload
```

Visibility: default

Description: `BytesOf` returns a copy of the string's content as an array of bytes. For an empty string, zero bytes are returned (i.e. `length(BytesOf(S))=0`).

See also: TBytes (1653)

76.15.40 ByteToCharIndex

Synopsis: Convert a character index in Bytes to an Index in characters.

Declaration:
```
function ByteToCharIndex(const S: string; Index: SizeInt) : SizeInt
```

Visibility: default

Description: `ByteToCharIndex` returns the index (in characters) of the `Index`-th byte in `S`.

Errors: This function does not take into account MBCS yet.

See also: CharToByteLen (1688), ByteToCharLen (1686)

76.15.41 ByteToCharLen

Synopsis: Convert a length in bytes to a length in characters.

Declaration:
```
function ByteToCharLen(const S: string; MaxLen: SizeInt) : SizeInt
```

Visibility: default

Description: `ByteToCharLen` returns the number of bytes in `S`, but limits the result to `MaxLen`

Errors: This function does not take into account MBCS yet.

See also: CharToByteLen (1688), ByteToCharIndex (1686)

76.15.42 ByteType

Synopsis: Return the type of byte in an ansistring for a multi-byte character set.

Declaration:
```
function ByteType(const S: string; Index: SizeUInt) : TMbcsByteType
```

Visibility: default

Description: `ByteType` returns the type of byte in the ansistring `S` at (1-based) position `Index`.

Errors: No checking on the index is performed.

See also: TMbcsByteType (1659), StrByteType (1773)

76.15.43 CallTerminateProcs

Synopsis: Call the exit chain procedures.

Declaration:
```
function CallTerminateProcs : Boolean
```

Visibility: default

Description: `CallTerminateProcs` is run on program exit. It executes all terminate procedures that were added to the exit chain with AddTerminateProc (1669), and does this in reverse order.

Errors: If one of the exit procedure raises an exception, it is *not* caught, and the remaining exit procedures will not be executed.

See also: TTerminateProc (1662), AddTerminateProc (1669)

76.15.44 ChangeFileExt

Synopsis: Change the extension of a filename.

Declaration:
```
function ChangeFileExt(const FileName: UnicodeString;
                       const Extension: UnicodeString) : UnicodeString
function ChangeFileExt(const FileName: RawByteString;
                       const Extension: RawByteString) : RawByteString
```

Visibility: default

Description: `ChangeFileExt` changes the file extension in `FileName` to `Extension`. The extension `Extension` includes the starting `.` (dot). The previous extension of `FileName` are all characters after the last `.`, the `.` character included.

If `FileName` doesn't have an extension, `Extension` is just appended.

Errors: None.

See also: ExtractFileExt (1711), ExtractFileName (1712), ExtractFilePath (1712), ExpandFileName (1708)

76.15.45 CharInSet

Synopsis: Check whether a char is in a set of characters.

Declaration:
```
function CharInSet(Ch: AnsiChar; const CSet: TSysCharSet) : Boolean
function CharInSet(Ch: WideChar; const CSet: TSysCharSet) : Boolean
```

Visibility: default

Description: `CharInSet` returns `True` if `Ch` matches one of the characters in `CSet`, it returns `False` otherwise. It is equivalent to

```
Ch in CSet
```

Later versions of this function may take WideChar into account.

76.15.46 CharToByteLen

Synopsis: Convert a length in characters to a length in bytes.

Declaration: `function CharToByteLen(const S: string; MaxLen: SizeInt) : SizeInt`

Visibility: default

Description: `CharToByteLen` returns the number of bytes in `S`, but limits the result to `MaxLen`

Errors: This function does not take into account MBCS yet.

See also: ByteToCharLen (1686), ByteToCharIndex (1686)

76.15.47 CheckOSError

Synopsis: Check for OS error and raise exception.

Declaration: `procedure CheckOSError(LastError: Integer)`

Visibility: default

Description: `CheckOSError` checks if `LastError` is non-zero and calls RaiseLastOSError (1766) if this is the case.

See also: RaiseLastOSError (1766)

76.15.48 CodePageNameToCodePage

Synopsis: Return a numeric identifier for the codepage.

Declaration:
```
function CodePageNameToCodePage(const cpname: AnsiString)
                                      : TSystemCodePage
```

Visibility: default

Description: `CodePageNameToCodePage` returns the code page number for the specified codepage `cpname`.

Errors: If the code page is not found in the list of code pages, `$FFFF` is returned.

See also: CodePageToCodePageName (1688)

76.15.49 CodePageToCodePageName

Synopsis: Convert a numeric codepage identifier to a codepage name.

Declaration: `function CodePageToCodePageName(cp: TSystemCodePage) : AnsiString`

Visibility: default

Description: `CodePageToCodePageName` returns the name of the codepage `cp`.

Errors: If no matching codepage is found in the list of codepages, an empty string is returned.

See also: CodePageNameToCodePage (1688)

76.15.50 CompareMem

Synopsis: Compare two memory areas.

Declaration:
```
function CompareMem(P1: Pointer; P2: Pointer; &Length: PtrUInt)
                          : Boolean
```

Visibility: default

Description: `CompareMem` compares, byte by byte, 2 memory areas pointed to by `P1` and `P2`, for a length of `L` bytes.

The function returns `True` if all `L` bytes are the same, and `False` otherwise.

76.15.51 CompareMemRange

Synopsis: Compare 2 memory locations.

Declaration:
```
function CompareMemRange(P1: Pointer; P2: Pointer; &Length: PtrUInt)
                                : Integer
```

Visibility: default

Description: `CompareMemRange` compares the 2 memory locations pointed to by `P1` and `P2` byte per byte. It stops comparing after `Length` bytes have been compared, or when it has encountered 2 different bytes. The result is then

>0if a byte in range `P1` was found that is bigger than the corresponding byte in range `P2`.

0if all bytes in range `P1` are the same as the corresponding bytes in range `P2`.

<0if a byte in range `P1` was found that is less than the corresponding byte in range `P2`.

Errors: None.

See also: SameText (1769)

76.15.52 CompareStr

Synopsis: Compare 2 ansistrings case-sensitively, ignoring special characters.

Declaration:
```
function CompareStr(const S1: string; const S2: string) : Integer
                    ;  Overload
function CompareStr(const S1: string; const S2: string;
                    LocaleOptions: TLocaleOptions) : Integer;  Overload
```

Visibility: default

Description: `CompareStr` compares two strings, `S1` and `S2`, and returns the following result:

<0if `S1<S2`.

0if `S1=S2`.

>0if `S1>S2`.

The comparison of the two strings is case-sensitive. The function does not take internationalization settings into account, it simply compares ASCII values.

Errors: None.

See also: AnsiCompareText (1671), AnsiCompareStr (1670), CompareText (1690)

Listing: ./examples/sysutex/ex65.pp

```
Program Example65;

{ This program demonstrates the CompareStr function }
{$H+}

Uses sysutils;

Procedure TestIt (S1,S2 : String);

Var R : Longint;

begin
  R:=CompareStr(S1,S2);
  Write ('"',S1,'" is ');
  If R<0 then
    write ('less than ')
  else If R=0 then
    Write ('equal to ')
  else
    Write ('larger than ');
  WriteIn ('"',S2,'"');
end;

Begin
  Testit('One string','One smaller string');
  Testit('One string','one string');
  Testit('One string','One string');
  Testit('One string','One tall string');
End.
```

76.15.53 CompareText

Synopsis: Compare 2 ansistrings case insensitive.

Declaration:
```
function CompareText(const S1: string; const S2: string) : Integer
                     ;  Overload
function CompareText(const S1: string; const S2: string;
                     LocaleOptions: TLocaleOptions) : Integer;  Overload
```

Visibility: default

Description: `CompareText` compares two strings, `S1` and `S2`, and returns the following result:

<0if `S1<S2`.

0if `S1=S2`.

>0if `S1>S2`.

The comparison of the two strings is case-insensitive. The function does not take internationalization settings into account, it simply compares ASCII values.

Errors: None.

See also: AnsiCompareText (1671), AnsiCompareStr (1670), CompareStr (1689)

Listing: ./examples/sysutex/ex66.pp

```
Program Example66;

{ This program demonstrates the CompareText function }
{$H+}

Uses sysutils;

Procedure TestIt (S1,S2 : String);

Var R : Longint;

begin
  R:=CompareText(S1,S2);
  Write ('"',S1,'" is ');
  If R<0 then
    write ('less than ')
  else If R=0 then
    Write ('equal to ')
  else
    Write ('larger than ');
  WriteIn ('"',S2,'"');
end;

Begin
  Testit('One string','One smaller string');
  Testit('One string','one string');
  Testit('One string','One string');
  Testit('One string','One tall string');
End.
```

76.15.54 ComposeDateTime

Synopsis: Add a date and time.

Declaration: `function ComposeDateTime(Date: TDateTime; Time: TDateTime) : TDateTime`

Visibility: default

Description: `ComposeDateTime` correctly adds `Date` and `Time`, also for dates before 1899-12-31. For dates after this date, it is just the mathematical addition.

Errors: None.

See also: #rtl.dateutils.EncodeDateTime (616)

76.15.55 ConcatPaths

Synopsis: Concatenate an array of paths to form a single path.

Declaration:
```
function ConcatPaths(const Paths: Array of UnicodeString)
                     : UnicodeString
function ConcatPaths(const Paths: Array of RawByteString)
                     : RawByteString
```

Visibility: default

Description: `ConcatPaths` will concatenate the different path components in `Paths` to a single path. It will insert directory separators between the various components of the path as needed. No directory separators will be added to the beginning or the end of the path, and none will be taken away.

See also: IncludeTrailingPathDelimiter (1756), IncludeLeadingPathDelimiter (1755), ExcludeTrailingPathDelimiter (1707), IncludeTrailingPathDelimiter (1756)

Listing: ./examples/sysutex/ex96.pp

```
program ex96;

{ This program demonstrates the Concatpaths function }

uses sysutils;

begin
  // will write /this/path/more/levels/
  Writeln(ConcatPaths(['/this/','path','more/levels/']));
  // will write this/path/more/levels/
  Writeln(ConcatPaths(['this/','path','more/levels/']));
  // will write this/path/more/levels
  Writeln(ConcatPaths(['this/','path','more/levels']));
end.
```

76.15.56 CreateDir

Synopsis: Create a new directory.

Declaration:
```
function CreateDir(const NewDir: RawByteString) : Boolean
function CreateDir(const NewDir: UnicodeString) : Boolean
```

Visibility: default

Description: `CreateDir` creates a new directory with name `NewDir`. If the directory doesn't contain an absolute path, then the directory is created below the current working directory.

The function returns `True` if the directory was successfully created, `False` otherwise.

Errors: In case of an error, the function returns `False`.

See also: RemoveDir (1766)

Listing: ./examples/sysutex/ex26.pp

```
Program Example26;

{ This program demonstrates the CreateDir and RemoveDir functions }
{ Run this program twice in the same directory }

Uses sysutils;

Begin
  If Not DirectoryExists('NewDir') then
    If Not CreateDir ('NewDir') Then
      Writeln ('Failed to create directory !')
    else
      Writeln ('Created "NewDir" directory')
  Else
```

```
  If Not RemoveDir ('NewDir') Then
    WriteIn ('Failed to remove directory !')
  else
    WriteIn ('Removed "NewDir" directory');
End.
```

76.15.57 CreateGUID

Synopsis: Create a new GUID.

Declaration: `function CreateGUID(out GUID: TGuid) : Integer`

Visibility: default

Description: `CreateGUID` can be called to create a new GUID (Globally Unique Identifier) value. The function returns the new `GUID` value in `GUID` and returns zero in case the GUID was created successfully. If no GUID was created, a nonzero error code is returned.

The default mechanism for creating a new GUID is system dependent. If operating system support is available, it is used. If none is available, a default implementation using random numbers is used.

The `OnCreateGUID` callback can be set to hook a custom mechanism behind the `CreateGUID` function. This can be used to let the GUID be created by an external GUID creation library.

Errors: On error, a nonzero return value is returned.

See also: GUIDCase (1754), IsEqualGUID (1759), StringToGUID (1779), TryStringToGUID (1806), GUIDToString (1754)

76.15.58 CurrentYear

Synopsis: Return the current year.

Declaration: `function CurrentYear : Word`

Visibility: default

Description: `CurrentYear` returns the current year as a 4-digit number.

Errors: None.

See also: Date (1694), Time (1801), Now (1765)

76.15.59 CurrToStr

Synopsis: Convert a currency value to a string.

Declaration:
```
function CurrToStr(Value: Currency) : string
function CurrToStr(Value: Currency;
                   const FormatSettings: TFormatSettings) : string
```

Visibility: default

Description: `CurrToStr` will convert a currency value to a string with a maximum of 15 digits, and precision 2. Calling `CurrToStr` is equivalent to calling FloatToStrF (1731):

```
FloatToStrF(Value,ffNumber,15,2);
```

Note that on unix systems, the localization support must be enabled explicitly, see Localization (1636).

Errors: None.

See also: FloatToStrF (1731), StrToCurr (1788)

76.15.60 CurrToStrF

Synopsis: Format a currency to a string.

Declaration:
```
function CurrToStrF(Value: Currency; Format: TFloatFormat;
                    Digits: Integer) : string
function CurrToStrF(Value: Currency; Format: TFloatFormat;
                    Digits: Integer;
                    const FormatSettings: TFormatSettings) : string
```

Visibility: default

Description: `CurrToStrF` formats the currency `Value` according to the value in `Format`, using the number of digits specified in `Digits`, and a precision of 19. This function simply calls FloatToStrF (1731).

Note that on unix systems, the localization support must be enabled explicitly, see Localization (1636).

See also: FloatToStrF (1731), Localization (1636)

76.15.61 Date

Synopsis: Return the current date.

Declaration:
```
function Date : TDateTime
```

Visibility: default

Description: `Date` returns the current date in `TDateTime` format.

Errors: None.

See also: Time (1801), Now (1765)

Listing: ./examples/sysutex/ex1.pp

```
Program Example1;

{ This program demonstrates the Date function }

uses sysutils;

Var YY,MM,DD : Word;

Begin
 Writeln ('Date : ',Date);
 DeCodeDate (Date,YY,MM,DD);
 Writeln (format ('Date is (DD/MM/YY): %d/%d/%d ',[dd,mm,yy]));
End.
```

76.15.62 DateTimeToFileDate

Synopsis: Convert a `TDateTime` value to a file age (integer).

Declaration:
```
function DateTimeToFileDate(DateTime: TDateTime) : Int64
```

Visibility: default

Description: `DateTimeToFileDate` function converts a date/time indication in `TDateTime` format to a file-date function, such as returned for instance by the FileAge (1714) function.

Errors: None.

See also: Time (1801), Date (1694), FileDateToDateTime (1716), DateTimeToSystemTime (1697), DateTimeToTimeStamp (1697)

Listing: ./examples/sysutex/ex2.pp

```
Program Example2;

{ This program demonstrates the DateTimeToFileDate function }

Uses sysutils;

Begin
  WriteIn ('FileTime of now would be: ',DateTimeToFileDate (Now));
End.
```

76.15.63 DateTimeToStr

Synopsis: Converts a `TDateTime` value to a string using a predefined format.

Declaration:
```
function DateTimeToStr(DateTime: TDateTime; ForceTimeIfZero: Boolean)
                        : string
function DateTimeToStr(DateTime: TDateTime;
                       const FormatSettings: TFormatSettings;
                       ForceTimeIfZero: Boolean) : string
```

Visibility: default

Description: `DateTimeToStr` returns a string representation of `DateTime` using the formatting specified in `ShortDateFormat` and `LongTimeFormat`. It corresponds to a call to `FormatDateTime('c',DateTime)` (see formatchars (1641)).

Note that if the time part is 0 (i.e. midnight), no time is appended.

Note that on unix systems, the localization support must be enabled explicitly, see Localization (1636).

Errors: None.

See also: FormatDateTime (1743), DefaultFormatSettings (1664), Localization (1636)

Listing: ./examples/sysutex/ex3.pp

```
Program Example3;

{ This program demonstrates the DateTimeToStr function }

Uses sysutils;
```

```
Begin
  WriteLn ('Today is : ',DateTimeToStr(Now));
  WriteLn ('Today is : ',FormatDateTime('c',Now));
End.
```

76.15.64 DateTimeToString

Synopsis: Converts a `TDateTime` value to a string with a given format.

Declaration:
```
procedure DateTimeToString(out Result: string; const FormatStr: string;
                           const DateTime: TDateTime;
                           Options: TFormatDateTimeOptions)
procedure DateTimeToString(out Result: string; const FormatStr: string;
                           const DateTime: TDateTime;
                           const FormatSettings: TFormatSettings;
                           Options: TFormatDateTimeOptions)
```

Visibility: default

Description: `DateTimeToString` returns in `Result` a string representation of `DateTime` using the formatting specified in `FormatStr`.

for a list of characters that can be used in the `FormatStr` formatting string, see formatchars (1641).

Note that for 'c', if the time part is 0 (i.e. midnight), no time is appended.

Note that on unix systems, the localization support must be enabled explicitly, see Localization (1636).

Errors: In case a wrong formatting character is found, an EConvertError (1830) is raised.

See also: FormatDateTime (1743), formatchars (1641), EConvertError (1830), Localization (1636)

Listing: ./examples/sysutex/ex4.pp

```
Program Example4;

{ This program demonstrates the DateTimeToString function }

Uses sysutils;


Procedure today (Fmt : string);

Var S : AnsiString;

begin
  DateTimeToString (S,Fmt,Date);
  WriteLn (S);
end;

Procedure Now (Fmt : string);

Var S : AnsiString;

begin
  DateTimeToString (S,Fmt,Time);
  WriteLn (S);
```

```
end;

Begin
  Today ('"Today is "dddd dd mmmm y');
  Today ('"Today is "d mmm yy');
  Today ('"Today is "d/mmm/yy');
  Now ('''The time is ''am/pmh:n:s');
  Now ('''The time is ''hh:nn:ssam/pm');
  Now ('''The time is ''tt');
End.
```

76.15.65 DateTimeToSystemTime

Synopsis: Converts a `TDateTime` value to a systemtime structure.

Declaration:
```
procedure DateTimeToSystemTime(DateTime: TDateTime;
                               out SystemTime: TSystemTime)
```

Visibility: default

Description: `DateTimeToSystemTime` converts a date/time pair in `DateTime`, with `TDateTime` format to a system time `SystemTime`.

Errors: None.

See also: DateTimeToFileDate (1695), SystemTimeToDateTime (1799), DateTimeToTimeStamp (1697)

Listing: ./examples/sysutex/ex5.pp

```
Program Example5;

{ This program demonstrates the DateTimeToSystemTime function }

Uses sysutils;

Var ST : TSystemTime;

Begin
  DateTimeToSystemTime(Now,ST);
  With St do
    begin
    WriteIn ('Today is     ',year,'/',month,'/',Day);
    WriteIn ('The time is ',Hour,':',minute,':',Second,'.',MilliSecond);
    end;
End.
```

76.15.66 DateTimeToTimeStamp

Synopsis: Converts a `TDateTime` value to a TimeStamp structure.

Declaration:
```
function DateTimeToTimeStamp(DateTime: TDateTime) : TTimeStamp
```

Visibility: default

Description: `DateTimeToSystemTime` converts a date/time pair in `DateTime`, with `TDateTime` format to a `TTimeStamp` format.

Errors: None.

See also: DateTimeToFileDate (1695), SystemTimeToDateTime (1799), DateTimeToSystemTime (1697)

Listing: ./examples/sysutex/ex6.pp

```
Program Example6;

{ This program demonstrates the DateTimeToTimeStamp function }

Uses sysutils;

Var TS : TTimeStamp;

Begin
  TS:=DateTimeToTimeStamp (Now);
  With TS do
    begin
    WriteLn ('Now is ',time,' millisecond past midnight');
    WriteLn ('Today is ' ,Date,' days past 1/1/0001');
    end;
End.
```

76.15.67 DateToStr

Synopsis: Converts a `TDateTime` value to a date string with a predefined format.

Declaration:
```
function DateToStr(Date: TDateTime) : string
function DateToStr(Date: TDateTime;
                   const FormatSettings: TFormatSettings) : string
```

Visibility: default

Description: `DateToStr` converts `Date` to a string representation. It uses `ShortDateFormat` as it's formatting string. It is hence completely equivalent to a `FormatDateTime('ddddd', Date)`.

Note that on unix systems, the localization support must be enabled explicitly, see Localization (1636).

Errors: None.

See also: TimeToStr (1802), DateTimeToStr (1695), FormatDateTime (1743), StrToDate (1788), Localization (1636)

Listing: ./examples/sysutex/ex7.pp

```
Program Example7;

{ This program demonstrates the DateToStr function }

Uses sysutils;

Begin
  WriteLn(Format ('Today is: %s',[DateToStr(Date)]));
End.
```

76.15.68 DayOfWeek

Synopsis: Returns the day of the week.

Declaration: `function DayOfWeek(DateTime: TDateTime) : Integer`

Visibility: default

Description: `DayOfWeek` returns the day of the week from `DateTime`. `Sunday` is counted as day 1, `Saturday` is counted as day 7. The result of `DayOfWeek` can serve as an index to the `LongDayNames` constant array, to retrieve the name of the day.

Errors: None.

See also: Date (1694), DateToStr (1698)

Listing: ./examples/sysutex/ex8.pp

```
Program Example8;

{ This program demonstrates the DayOfWeek function }

Uses sysutils;

Begin
  Writeln ('Today''s day is ',LongDayNames[DayOfWeek(Date)]);
End.
```

76.15.69 DecodeDate

Synopsis: Decode a `TDateTime` to a year,month,day triplet.

Declaration: `procedure DecodeDate(Date: TDateTime; out Year: Word; out Month: Word; out Day: Word)`

Visibility: default

Description: `DecodeDate` decodes the Year, Month and Day stored in `Date`, and returns them in the `Year`, `Month` and `Day` variables.

Errors: None.

See also: EncodeDate (1704), DecodeTime (1700)

Listing: ./examples/sysutex/ex9.pp

```
Program Example9;

{ This program demonstrates the DecodeDate function }

Uses sysutils;

Var YY,MM,DD : Word;

Begin
  DecodeDate(Date,YY,MM,DD);
  Writeln (Format ('Today is %d/%d/%d',[dd,mm,yy]));
End.
```

76.15.70 DecodeDateFully

Synopsis: Decode a date with additional date of the week.

Declaration:
```
function DecodeDateFully(const DateTime: TDateTime; out Year: Word;
                         out Month: Word; out Day: Word; out DOW: Word)
                          : Boolean
```

Visibility: default

Description: `DecodeDateFully`, like DecodeDate (1699), decodes `DateTime` in its parts and returns these in `Year`, `Month`, `Day` but in addition returns the day of the week in `DOW`.

Errors: None.

See also: EncodeDate (1704), TryEncodeDate (1805), DecodeDate (1699)

76.15.71 DecodeTime

Synopsis: Decode a `TDateTime` to a hour,minute,second,millisec quartet.

Declaration:
```
procedure DecodeTime(Time: TDateTime; out Hour: Word; out Minute: Word;
                     out Second: Word; out MilliSecond: Word)
```

Visibility: default

Description: `DecodeTime` decodes the hours, minutes, second and milliseconds stored in `Time`, and returns them in the `Hour`, `Minute` and `Second` and `MilliSecond` variables.

Errors: None.

See also: EncodeTime (1704), DecodeDate (1699)

Listing: ./examples/sysutex/ex10.pp

```
Program Example10;

{ This program demonstrates the DecodeTime function }

Uses sysutils;

Var HH,MM,SS,MS: Word;

Begin
  DecodeTime(Time,HH,MM,SS,MS);
  Writeln (format('The time is %d:%d:%d.%d',[hh,mm,ss,ms]));
End.
```

76.15.72 DeleteFile

Synopsis: Delete a file from the file system.

Declaration:
```
function DeleteFile(const FileName: UnicodeString) : Boolean
function DeleteFile(const FileName: RawByteString) : Boolean
```

Visibility: default

Description: `DeleteFile` deletes file `FileName` from disk. The function returns `True` if the file was successfully removed, `False` otherwise.

Errors: On error, `False` is returned.

See also: FileCreate (1715), FileExists (1717)

Listing: ./examples/sysutex/ex31.pp

```
Program Example31;

{ This program demonstrates the DeleteFile function }

Uses sysutils;

Var
  Line : String;
  F,I : Longint;

Begin
  F:=FileCreate('test.txt');
  Line:='Some string line.'#10;
  For I:=1 to 10 do
    FileWrite (F,Line[1],Length(Line));
  FileClose(F);
  DeleteFile('test.txt');
End.
```

76.15.73 DirectoryExists

Synopsis: Check whether a directory exists in the file system.

Declaration:
```
function DirectoryExists(const Directory: UnicodeString;
                         FollowLink: Boolean) : Boolean
function DirectoryExists(const Directory: RawByteString;
                         FollowLink: Boolean) : Boolean
```

Visibility: default

Description: `DirectoryExists` checks whether `Directory` exists in the file system and is actually a directory. If this is the case, the function returns `True`, otherwise `False` is returned.

See also: FileExists (1717)

76.15.74 DiskFree

Synopsis: Return the amount of free diskspace.

Declaration:
```
function DiskFree(drive: Byte) : Int64
```

Visibility: default

Description: `DiskFree` returns the free space (in bytes) on disk `Drive`. Drive is the number of the disk drive:

0 for the current drive.

1 for the first floppy drive.

2 for the second floppy drive.

3 for the first hard-disk partition.

4-26for all other drives and partitions.

Remark Under Linux, and Unix in general, the concept of disk is different than the dos one, since the file system is seen as one big directory tree. For this reason, the `DiskFree` and DiskSize (1702) functions must be mimicked using filenames that reside on the partitions. For more information, see AddDisk (1668).

Errors: On error, `-1` is returned.

See also: DiskSize (1702), AddDisk (1668)

Listing: ./examples/sysutex/ex27.pp

```
Program Example27;

{ This program demonstrates the DiskFree function }

Uses sysutils;

Begin
  Write ('Size of current disk       : ',DiskSize(0));
  Writeln (' (= ',DiskSize(0) div 1024,'k)');
  Write ('Free space of current disk : ',Diskfree(0));
  Writeln (' (= ',Diskfree(0) div 1024,'k)');
End.
```

76.15.75 DiskSize

Synopsis: Return the total amount of diskspace.

Declaration: `function DiskSize(drive: Byte) : Int64`

Visibility: default

Description: `DiskSize` returns the size (in bytes) of disk `Drive`. Drive is the number of the disk drive:

0for the current drive.

1for the first floppy drive.

2for the second floppy drive.

3for the first hard-disk partition.

4-26for all other drives and partitions.

Remark Under Linux, and Unix in general, the concept of disk is different than the dos one, since the file system is seen as one big directory tree. For this reason, the DiskFree (1701) and `DiskSize` functions must be mimicked using filenames that reside on the partitions. For more information, see AddDisk (1668)

For an example, see DiskFree (1701).

Errors: On error, `-1` is returned.

See also: DiskFree (1701), AddDisk (1668)

76.15.76 DisposeStr

Synopsis: Dispose an ansistring from the heap.

Declaration:
```
procedure DisposeStr(S: PString);  Overload
procedure DisposeStr(S: PShortString);  Overload
```

Visibility: default

Description: `DisposeStr` removes the dynamically allocated string `S` from the heap, and releases the occupied memory.

This function is provided for Delphi compatibility only. `AnsiStrings` are managed on the heap and should be preferred to the mechanism of dynamically allocated strings.

For an example, see DisposeStr (1703).

Errors: None.

See also: NewStr (1764), AppendStr (1683), AssignStr (1684)

76.15.77 DoDirSeparators

Synopsis: Convert known directory separators to the current directory separator.

Declaration:
```
procedure DoDirSeparators(var FileName: UnicodeString)
procedure DoDirSeparators(var FileName: RawByteString)
```

Visibility: default

Description: This function replaces all known directory separators in `FileName` to the directory separator character for the current system. The list of known separators is specified in the AllowDirectorySeparators (1369) constant.

Errors: None.

See also: ExtractFileName (1712), ExtractFilePath (1712)

Listing: ./examples/sysutex/ex32.pp

```
Program Example32;

{ This program demonstrates the DoDirSeparators function }
{$H+}

Uses sysutils;

Procedure Testit (F : String);

begin
  Writeln ('Before : ',F);
  DoDirSeparators (F);
  Writeln ('After  : ',F);
end;

Begin
  Testit (GetCurrentDir);
  Testit ('c:\pp\bin\win32');
  Testit ('/usr/lib/fpc');
  Testit ('\usr\lib\fpc');
End.
```

76.15.78 EncodeDate

Synopsis: Encode a Year,Month,Day to a `TDateTime` value.

Declaration:
```
function EncodeDate(Year: Word; Month: Word; Day: Word) : TDateTime
```

Visibility: default

Description: `EncodeDate` encodes the `Year`, `Month` and `Day` variables to a date in `TDateTime` format. It does the opposite of the DecodeDate (1699) procedure.

The parameters must lie withing valid ranges (boundaries included):

Yearmust be between 1 and 9999.

Monthmust be within the range 1-12.

Daymust be between 1 and 31.

Errors: In case one of the parameters is out of it's valid range, an EConvertError (1830) exception is raised.

See also: EncodeTime (1704), DecodeDate (1699)

Listing: ./examples/sysutex/ex11.pp

```
Program Example11;

{ This program demonstrates the EncodeDate function }

Uses sysutils;

Var YY,MM,DD : Word;

Begin
  DecodeDate (Date,YY,MM,DD);
  WriteLn ('Today is : ',FormatDateTime ('dd mmmm yyyy',EnCodeDate(YY,Mm,Dd)));
End.
```

76.15.79 EncodeTime

Synopsis: Encode a Hour,Min,Sec,millisec to a `TDateTime` value.

Declaration:
```
function EncodeTime(Hour: Word; Minute: Word; Second: Word;
                    MilliSecond: Word) : TDateTime
```

Visibility: default

Description: `EncodeTime` encodes the `Hour`, `Minute`, `Second`, `MilliSecond` variables to a `TDateTime` format result. It does the opposite of the DecodeTime (1700) procedure.

The parameters must have a valid range (boundaries included):

Hourmust be between 0 and 23.

Minute,secondmust both be between 0 and 59.

Millisecondmust be between 0 and 999.

Errors: In case one of the parameters is out of it's valid range, an EConvertError (1830) exception is raised.

See also: EncodeDate (1704), DecodeTime (1700)

Listing: ./examples/sysutex/ex12.pp

```
Program Example12;

{ This program demonstrates the EncodeTime function }

Uses sysutils;

Var Hh,MM,SS,MS : Word;

Begin
  DeCodeTime (Time,Hh,MM,SS,MS);
  WriteIn ('Present Time is : ',FormatDateTime('hh:mm:ss',EnCodeTime (HH,MM,SS,MS)));
End.
```

76.15.80 ExceptAddr

Synopsis: Current exception address.

Declaration: `function ExceptAddr : CodePointer`

Visibility: default

Description: `ExceptAddr` returns the address from the currently treated exception object when an exception is raised, and the stack is unwound.

See also: ExceptObject (1706), ExceptionErrorMessage (1706), ShowException (1770)

76.15.81 ExceptFrameCount

Synopsis: Number of frames included in an exception backtrace.

Declaration: `function ExceptFrameCount : LongInt`

Visibility: default

Description: `ExceptFrameCount` returns the number of frames that are included in an exception stack frame backtrace. The function returns 0 if there is currently no exception being handled. (i.e. it only makes sense to call this function in an `finally..end` or `except..end` block.

Errors: None.

See also: ExceptFrames (1705), ExceptAddr (1705), ExceptObject (1706), #rtl.system.ExceptProc (1372)

76.15.82 ExceptFrames

Synopsis: Return the current exception stack frames.

Declaration: `function ExceptFrames : PCodePointer`

Visibility: default

Description: `ExceptFrames` returns the current list frames on the exception stack. If there is no exception in progress, `Nil` is returned.

See also: ExceptFrameCount (1705), ExceptAddr (1705), ExceptObject (1706), #rtl.system.ExceptProc (1372)

76.15.83 ExceptionErrorMessage

Synopsis: Return a message describing the exception.

Declaration:
```
function ExceptionErrorMessage(ExceptObject: TObject;
                               ExceptAddr: Pointer; Buffer: PChar;
                               Size: Integer) : Integer
```

Visibility: default

Description: `ExceptionErrorMessage` creates a string that describes the exception object `ExceptObject` at address `ExceptAddr`. It can be used to display exception messages. The string will be stored in the memory pointed to by `Buffer`, and will at most have `Size` characters.

The routine checks whether `ExceptObject` is a Exception (1839) object or not, and adapts the output accordingly.

See also: ExceptObject (1706), ExceptAddr (1705), ShowException (1770)

76.15.84 ExceptObject

Synopsis: Current Exception object.

Declaration:
```
function ExceptObject : TObject
```

Visibility: default

Description: `ExceptObject` returns the currently treated exception object when an exception is raised, and the stack is unwound.

Errors: If there is no exception, the function returns `Nil`

See also: ExceptAddr (1705), ExceptionErrorMessage (1706), ShowException (1770)

76.15.85 ExcludeLeadingPathDelimiter

Synopsis: Strip the leading path delimiter of a path.

Declaration:
```
function ExcludeLeadingPathDelimiter(const Path: UnicodeString)
                                     : UnicodeString
function ExcludeLeadingPathDelimiter(const Path: RawByteString)
                                     : RawByteString
```

Visibility: default

Description: `ExcludeLeadingPathDelimiter` will remove any path delimiter on the first position of `Path` if there is one. if there is none (or the path is empty), it is left untouched.

See also: IncludeTrailingPathDelimiter (1756), IncludeLeadingPathDelimiter (1755), ExcludeTrailingPathDelimiter (1707), ConcatPaths (1691)

Listing: ./examples/sysutex/ex95.pp

```
Program Example95;

{ This program demonstrates the IncludeLeadingPathDelimiter function }

Uses sysutils;
```

```
Begin
  // Will print "/this/path"
  WriteLn(IncludeLeadingPathDelimiter('this/path'));
  // The same result
  WriteLn(IncludeLeadingPathDelimiter('/this/path'));
End.
```

76.15.86 ExcludeTrailingBackslash

Synopsis: Strip trailing directory separator from a pathname, if needed.

Declaration:
```
function ExcludeTrailingBackslash(const Path: UnicodeString)
                                  : UnicodeString
function ExcludeTrailingBackslash(const Path: RawByteString)
                                  : RawByteString
```

Visibility: default

Description: `ExcludeTrailingBackslash` is provided for backwards compatibility with Delphi. Use ExcludeTrailingPathDelimiter (1707) instead.

See also: IncludeTrailingPathDelimiter (1756), ExcludeTrailingPathDelimiter (1707), PathDelim (1648), IsPathDelimiter (1761)

76.15.87 ExcludeTrailingPathDelimiter

Synopsis: Strip trailing directory separator from a pathname, if needed.

Declaration:
```
function ExcludeTrailingPathDelimiter(const Path: UnicodeString)
                                      : UnicodeString
function ExcludeTrailingPathDelimiter(const Path: RawByteString)
                                      : RawByteString
```

Visibility: default

Description: `ExcludeTrailingPathDelimiter` removes the trailing path delimiter character (PathDelim (1648)) from `Path` if it is present, and returns the result.

See also: ExcludeTrailingBackslash (1707), IncludeTrailingPathDelimiter (1756), PathDelim (1648), IsPathDelimiter (1761)

76.15.88 ExecuteProcess

Synopsis: Execute another process (program).

Declaration:
```
function ExecuteProcess(const Path: RawByteString;
                        const ComLine: RawByteString;
                        Flags: TExecuteFlags) : Integer
function ExecuteProcess(const Path: RawByteString;
                        const ComLine: Array of RawByteString;
                        Flags: TExecuteFlags) : Integer
function ExecuteProcess(const Path: UnicodeString;
                        const ComLine: UnicodeString;
                        Flags: TExecuteFlags) : Integer
function ExecuteProcess(const Path: UnicodeString;
```

```
                                const ComLine: Array of UnicodeString;
                                Flags: TExecuteFlags) : Integer
```

Visibility: default

Description: `ExecuteProcess` will execute the program in `Path`, passing it the arguments in `ComLine`. `ExecuteProcess` will then wait for the program to finish, and will return the exit code of the executed program. In case `ComLine` is a single string, it will be split out in an array of strings, taking into account common whitespace and quote rules.

The program specified in `Path` is not searched in the searchpath specified in the `PATH` environment variable, so the full path to the executable must be specified in `Path`, although some operating systems may perform this search anyway (notably, windows)

`Flags` can be used to control the passing of file handles: if `ExecInheritsHandles` is included, the file handles of the current process will be passed on to the newly executed process.

Errors: In case the program could not be executed or an other error occurs, an EOSError (1836) exception will be raised.

See also: TExecuteFlags (1654), EOSError (1836)

76.15.89 ExeSearch

Synopsis: Search for an executable.

Declaration:
```
function ExeSearch(const Name: UnicodeString;
                   const DirList: UnicodeString) : UnicodeString
function ExeSearch(const Name: RawByteString;
                   const DirList: RawByteString) : RawByteString
```

Visibility: default

Description: `ExeSearch` searches for an executable `Name` in the list of directories `DirList` (a list of directories, separator by PathSeparator (1386). If the current OS also searches implicitly in the current working directory, the current directory is searched in the first place.

If the executable is found, then the full path of the executable is returned. If it is not found, an empty string is returned.

No check is performed whether the found file is actually executable.

See also: FileSearch (1722)

76.15.90 ExpandFileName

Synopsis: Expand a relative filename to an absolute filename.

Declaration:
```
function ExpandFileName(const FileName: UnicodeString) : UnicodeString
function ExpandFileName(const FileName: RawByteString) : RawByteString
```

Visibility: default

Description: `ExpandFileName` expands the filename to an absolute filename. It changes all directory separator characters to the one appropriate for the system first.

If an empty filename is passed, it is expanded to the current directory.

If BasePath is not passed, the current directory is used as base path.

Errors: None.

See also: ExpandFileNameCase (1709), ExtractFileName (1712), ExtractFilePath (1712), ExtractFileDir (1710), ExtractFileDrive (1711), ExtractFileExt (1711), ExtractRelativePath (1712)

Listing: ./examples/sysutex/ex33.pp

```
Program Example33;

{ This program demonstrates the ExpandFileName function }

Uses sysutils;

Procedure Testit (F : String);

begin
  WriteIn (F,' expands to : ',ExpandFileName(F));
end;

Begin
  Testit('ex33.pp');
  Testit(ParamStr(0));
  Testit('/pp/bin/win32/ppc386');
  Testit('\pp\bin\win32\ppc386');
  Testit('.');
End.
```

76.15.91 ExpandFileNameCase

Synopsis: Expand a filename entered as case insensitive to the full path as stored on the disk.

Declaration:
```
function ExpandFileNameCase(const FileName: UnicodeString;
                            out MatchFound: TFilenameCaseMatch)
                             : UnicodeString
function ExpandFileNameCase(const FileName: RawByteString;
                            out MatchFound: TFilenameCaseMatch)
                             : RawByteString
```

Visibility: default

Description: On case insensitive platforms, `ExpandFileNameCase` behaves similarly to ExpandFileName (1708) except for the fact that it returns the final part of the path with the same case of letters as found on the disk (if it exists - otherwise the case equals the one provided on input). On case sensitive platforms it also checks whether one or more full paths exist on disk which would correspond to the provided input if treated case insensitively and returns the first such match found and information whether the match is unique or not.

Note that the behaviour is basically undefined if the input includes wildcards characters. Normally, wildcards in the last part of path provided on input are resolved to the first corresponding item found on the disk, but it is better not to rely on that and use other more suitable functions if working with wildcards like FindFirst (1727)/FindNext (1728).

Errors: None.

See also: ExpandFileName (1708), ExtractFileName (1712), ExtractFilePath (1712), ExtractFileDir (1710), ExtractFileDrive (1711), ExtractFileExt (1711), ExtractRelativePath (1712)

Listing: ./examples/sysutex/ex33.pp

```
Program Example33;

{ This program demonstrates the ExpandFileName function }

Uses sysutils;

Procedure Testit (F : String);

begin
  WriteIn (F,' expands to : ',ExpandFileName(F));
end;

Begin
  Testit('ex33.pp');
  Testit(ParamStr(0));
  Testit('/pp/bin/win32/ppc386');
  Testit('\pp\bin\win32\ppc386');
  Testit('.');
End.
```

76.15.92 ExpandUNCFileName

Synopsis: Expand a relative filename to an absolute UNC filename.

Declaration:
```
function ExpandUNCFileName(const FileName: UnicodeString)
                           : UnicodeString
function ExpandUNCFileName(const FileName: RawByteString)
                           : RawByteString
```

Visibility: default

Description: `ExpandUNCFileName` runs ExpandFileName (1708) on `FileName` and then attempts to replace the drive letter by the name of a shared disk.

Errors: If an unexpected error occurs while determining the name of the shared disk, an empty string is returned.

See also: ExpandFileName (1708), ExtractFileName (1712), ExtractFilePath (1712), ExtractFileDir (1710), ExtractFileDrive (1711), ExtractFileExt (1711), ExtractRelativePath (1712)

76.15.93 ExtractFileDir

Synopsis: Extract the drive and directory part of a filename.

Declaration:
```
function ExtractFileDir(const FileName: UnicodeString) : UnicodeString
function ExtractFileDir(const FileName: RawByteString) : RawByteString
```

Visibility: default

Description: `ExtractFileDir` returns only the directory part of `FileName`, including a drive letter. The directory name has NO ending directory separator, in difference with ExtractFilePath (1712).

Errors: None.

See also: ExtractFileName (1712), ExtractFilePath (1712), ExtractFileDir (1710), ExtractFileDrive (1711), ExtractFileExt (1711), ExtractRelativePath (1712)

Listing: ./examples/sysutex/ex34.pp

```
Program Example34;

{ This program demonstrates the ExtractFileName function }
{$H+}
Uses sysutils;

Procedure Testit(F : String);

begin
  WriteIn ('FileName       : ',F);
  WriteIn ('Has Name       : ',ExtractFileName(F));
  WriteIn ('Has Path       : ',ExtractFilePath(F));
  WriteIn ('Has Extension  : ',ExtractFileExt(F));
  WriteIn ('Has Directory  : ',ExtractFileDir(F));
  WriteIn ('Has Drive      : ',ExtractFileDrive(F));
end;

Begin
   Testit (Paramstr(0));
   Testit ('/usr/local/bin/mysqld');
   Testit ('c:\pp\bin\win32\ppc386.exe');
   Testit ('/pp/bin/win32/ppc386.exe');
End.
```

76.15.94 ExtractFileDrive

Synopsis: Extract the drive part from a filename.

Declaration:
```
function ExtractFileDrive(const FileName: UnicodeString) : UnicodeString
function ExtractFileDrive(const FileName: RawByteString) : RawByteString
```

Visibility: default

Description: `Extracts` the drive letter from a filename. Note that some operating systems do not support drive letters.

For an example, see ExtractFileDir (1710).

See also: ExtractFileName (1712), ExtractFilePath (1712), ExtractFileDir (1710), ExtractFileDrive (1711), ExtractFileExt (1711), ExtractRelativePath (1712)

76.15.95 ExtractFileExt

Synopsis: Return the extension from a filename.

Declaration:
```
function ExtractFileExt(const FileName: UnicodeString) : UnicodeString
function ExtractFileExt(const FileName: RawByteString) : RawByteString
```

Visibility: default

Description: `ExtractFileExt` returns the extension (including the `.`(dot) character) of `FileName`.

For an example, see ExtractFileDir (1710).

Errors: None.

See also: ChangeFileExt (1687), ExtractFileName (1712), ExtractFilePath (1712), ExtractFileDir (1710), ExtractFileDrive (1711), ExtractFileExt (1711), ExtractRelativePath (1712)

76.15.96 ExtractFileName

Synopsis: Extract the filename part from a full path filename.

Declaration:
```
function ExtractFileName(const FileName: UnicodeString) : UnicodeString
function ExtractFileName(const FileName: RawByteString) : RawByteString
```

Visibility: default

Description: `ExtractFileName` returns the filename part from `FileName`. The filename consists of all characters after the last directory separator character ('/' or '\') or drive letter.

The full filename can always be reconstructed by concatenating the result of ExtractFilePath (1712) and `ExtractFileName`.

For an example, see ExtractFileDir (1710).

Errors: None.

See also: ExtractFileName (1712), ExtractFilePath (1712), ExtractFileDir (1710), ExtractFileDrive (1711), ExtractFileExt (1711), ExtractRelativePath (1712)

76.15.97 ExtractFilePath

Synopsis: Extract the path from a filename.

Declaration:
```
function ExtractFilePath(const FileName: UnicodeString) : UnicodeString
function ExtractFilePath(const FileName: RawByteString) : RawByteString
```

Visibility: default

Description: `ExtractFilePath` returns the path part (including drive letter) from `FileName`. The path consists of all characters before the last directory separator character ('/' or '\'), including the directory separator itself. In case there is only a drive letter, that will be returned.

The full filename can always be reconstructed by concatenating the result of `ExtractFilePath` and ExtractFileName (1712).

For an example, see ExtractFileDir (1710).

Errors: None.

See also: ExtractFileName (1712), ExtractFilePath (1712), ExtractFileDir (1710), ExtractFileDrive (1711), ExtractFileExt (1711), ExtractRelativePath (1712)

76.15.98 ExtractRelativePath

Synopsis: Extract a relative path from a filename, given a base directory.

Declaration:
```
function ExtractRelativePath(const BaseName: UnicodeString;
                             const DestName: UnicodeString)
                              : UnicodeString
function ExtractRelativePath(const BaseName: RawByteString;
                             const DestName: RawByteString)
                              : RawByteString
```

Visibility: default

Description: `ExtractRelativePath` constructs a relative path to go from `BaseName` to `DestName`. If `DestName` is on another drive (Not on Unix-like platforms) then the whole `Destname` is returned.

Note that directories must end on a path delimiter for this function to work correctly. If not, the last part is stripped and treated as a file name.

Errors: None.

See also: ExtractFileName (1712), ExtractFilePath (1712), ExtractFileDir (1710), ExtractFileDrive (1711), ExtractFileExt (1711)

Listing: ./examples/sysutex/ex35.pp

```
Program Example35;

{ This program demonstrates the ExtractRelativePath function }

Uses sysutils;

Procedure Testit (FromDir,ToDir : String);

begin
  Write ('From "',FromDir,'" to "',ToDir,'" via "');
  WriteIn (ExtractRelativePath(FromDir,ToDir),'"');
end;

Begin
 Testit ('/pp/src/compiler/','/pp/bin/win32/ppc386/');
 Testit ('/pp/bin/win32/ppc386/','/pp/src/compiler/');
 Testit ('/pp/bin/win32/','/pp/src/compiler/ppcx386/');
 Testit ('/pp/bin/win32/','/pp/src/compiler/ppcx386');
 Testit ('/pp/bin/win32','/pp/src/compiler/ppcx386');
 Testit ('e:/pp/bin/win32/ppc386/','d:/pp/src/compiler/');
 Testit ('e:\pp\bin\win32\ppc386/','d:\pp\src\compiler/');
 Testit ('C:\FPC\3.0.2\','C:\FPC\3.0.2\');
 Testit ('C:\FPC\3.0.2\','C:\FPC\3.0.4rc1\');
 Testit ('Q:\','Q:\FPC\3.0.4rc1\');
End.
```

76.15.99 ExtractShortPathName

Synopsis: Returns a 8.3 path name.

Declaration:
```
function ExtractShortPathName(const FileName: UnicodeString)
                                  : UnicodeString
function ExtractShortPathName(const FileName: RawByteString)
                                  : RawByteString
```

Visibility: default

Description: `ExtractShortPathName` returns a 8.3 compliant filename that represents the same file as `FileName`. On platforms other than windows, this is `FileName` itself.

See also: ExtractFilePath (1712), ExtractFileName (1712)

76.15.100 FileAge

Synopsis: Return the timestamp of a file.

Declaration:
```
function FileAge(const FileName: UnicodeString) : Int64
function FileAge(const FileName: UnicodeString;
                 out FileDateTime: TDateTime; FollowLink: Boolean)
                  : Boolean
function FileAge(const FileName: RawByteString;
                 out FileDateTime: TDateTime; FollowLink: Boolean)
                  : Boolean
function FileAge(const FileName: RawByteString) : Int64
```

Visibility: default

Description: `FileAge` in it's simplest form returns the last modification time of file `FileName` as an opaque os-dependent file timestamp. In this case, the return value can be transformed to a `TDateTime` value with the FileDateToDateTime (1716) function.

In case a `FileDateTime` argument is supplied, the `FileAge` function will do the transformation by itself, and the file time will be returned in the `FileDateTime` argument as a `TDateTime` value. The function return value is then a boolean indicating success or failure.

The `FollowSymlink` parameter can be set to `False` to read the timestamp of a symbolic link, instead of the timestamp of the file the symbolic link points to (which is the default behaviour).

FileAge cannot be used on directories, it will return -1 (or `False`) if `FileName` indicates a directory.

Errors: In case of errors, `-1` or `False` is returned.

See also: FileDateToDateTime (1716), FileExists (1717), FileGetAttr (1718)

Listing: ./examples/sysutex/ex36.pp

```
Program Example36;

{ This program demonstrates the FileAge function }

Uses sysutils;

Var S : TDateTime;
    fa : Longint;
Begin
  fa:=FileAge('ex36.pp');
  If Fa<>-1 then
    begin
    S:=FileDateTodateTime(fa);
    WriteIn ('I''m from ',DateTimeToStr(S))
    end;
End.
```

76.15.101 FileAgeUTC

Declaration:
```
function FileAgeUTC(const FileName: UnicodeString;
                    out FileDateTimeUTC: TDateTime; FollowLink: Boolean)
                     : Boolean
function FileAgeUTC(const FileName: RawByteString;
                    out FileDateTimeUTC: TDateTime; FollowLink: Boolean)
                     : Boolean
```

Visibility: default

76.15.102 FileClose

Synopsis: Close a file handle.

Declaration:
```
procedure FileClose(Handle: THandle)
```

Visibility: default

Description: `FileClose` closes the file handle `Handle`. After this call, attempting to read or write from the handle will result in an error.

For an example, see FileCreate (1715)

Errors: None.

See also: FileCreate (1715), FileWrite (1726), FileOpen (1721), FileRead (1722), FileTruncate (1725), FileSeek (1723)

76.15.103 FileCreate

Synopsis: Create a new file and return a handle to it.

Declaration:
```
function FileCreate(const FileName: UnicodeString) : THandle
function FileCreate(const FileName: UnicodeString; Rights: Integer)
                    : THandle
function FileCreate(const FileName: UnicodeString; ShareMode: Integer;
                    Rights: Integer) : THandle
function FileCreate(const FileName: RawByteString) : THandle
function FileCreate(const FileName: RawByteString; Rights: Integer)
                    : THandle
function FileCreate(const FileName: RawByteString; ShareMode: Integer;
                    Rights: Integer) : THandle
```

Visibility: default

Description: `FileCreate` creates a new file with name `FileName` on the disk and returns a file handle which can be used to read or write from the file with the FileRead (1722) and FileWrite (1726) functions.

If a file with name `FileName` already existed on the disk, it is overwritten.

The `ShareMode` parameter determines how file sharing is handled for the new file. It can have one or more of the following values, OR-ed together:

Table 76.24:

Name	Description
fmShareCompat	Open file in DOS share-compatibility mode.
fmShareDenyNone	Do not lock file.
fmShareDenyRead	Lock file so other processes cannot read.
fmShareDenyWrite	Lock file so other processes can only read.
fmShareExclusive	Lock file for exclusive use.

The optional `Rights` parameter only has an effect under UNIX, where it can be used to set the mode (read, write, execute, sticky bit, setgid and setuid flags) of the created file to the specified custom value. On other platforms, the `Rights` parameter is ignored.

Errors: If an error occurs (e.g. disk full or non-existent path), the function returns `THandle(-1)`.

See also: FileClose (1715), FileWrite (1726), FileOpen (1721), FileRead (1722), FileTruncate (1725), FileSeek (1723)

Listing: ./examples/sysutex/ex37.pp

```
Program Example37;

{ This program demonstrates the FileCreate/FileSeek/FileReed/FileTruncate functions }

Uses sysutils;

Var I,J,F : Longint;

Begin
  F:=FileCreate ('test.dat');
  If F=-1 then
    Halt(1);
  For I:=0 to 100 do
    FileWrite(F,I,SizeOf(i));
  FileClose(f);
  F:=FileOpen ('test.dat',fmOpenRead);
  For I:=0 to 100 do
    begin
    FileRead (F,J,SizeOF(J));
    If J<>I then
      WriteIn ('Mismatch at file position ',I)
    end;
  FileSeek(F,0,fsFromBeginning);
  Randomize;
  Repeat
    FileSeek(F,Random(100)*4,fsFromBeginning);
    FileRead (F,J,SizeOf(J));
    WriteIn ('Random read : ',j);
  Until J>80;
  FileClose(F);
  F:=FileOpen('test.dat',fmOpenWrite);
  I:=50*SizeOf(Longint);
  If FileTruncate(F,I) then
    WriteIn('SuccessFully truncated file to ',I,' bytes.');
  FileClose(F);
End.
```

76.15.104 FileDateToDateTime

Synopsis: Convert a `FileDate` value to a `TDateTime` value.

Declaration: `function FileDateToDateTime(Filedate: Int64) : TDateTime`

Visibility: default

Description: `FileDateToDateTime` converts the date/time encoded in `filedate` to a `TDateTime` encoded form. It can be used to convert date/time values returned by the FileAge (1714) or FindFirst (1727)/FindNext (1728) functions to `TDateTime` form.

Errors: None.

See also: DateTimeToFileDate (1695)

Listing: ./examples/sysutex/ex13.pp

```
Program Example13;

{ This program demonstrates the FileDateToDateTime function }

Uses sysutils;

Var
  ThisAge : Longint;

Begin
 Write ('ex13.pp created on :');
 ThisAge:=FileAge('ex13.pp');
 WriteIn (DateTimeToStr(FileDateToDateTime(ThisAge)));
End.
```

76.15.105 FileDateToUniversal

Declaration: `function FileDateToUniversal(Filedate: Int64) : TDateTime`

Visibility: default

76.15.106 FileExists

Synopsis: Check whether a particular file exists in the file system.

Declaration:
```
function FileExists(const FileName: UnicodeString; FollowLink: Boolean)
                    : Boolean
function FileExists(const FileName: RawByteString; FollowLink: Boolean)
                    : Boolean
```

Visibility: default

Description: `FileExists` returns `True` if a file with name `FileName` exists on the disk, `False` otherwise. On windows, this function will return `False` if a directory is passed as `FileName`.

On Unixes, passing a directory name used to result in `True`. (The rationale is that on UNIX, a directory is a file as well). As of version 3.2.0, this behaviour has been changed for a Delphi-compatible approach.

Note that this function accepts a single filename as an argument, without wildcards. To check for the existence of multiple files, see the FindFirst (1727) function.

Errors: None.

See also: FindFirst (1727), FileAge (1714), FileGetAttr (1718), FileSetAttr (1724)

Listing: ./examples/sysutex/ex38.pp

```
Program Example38;

{ This program demonstrates the FileExists function }

Uses sysutils;
```

```
Begin
  If FileExists(ParamStr(0)) Then
    WriteIn ('All is well, I seem to exist.');
End.
```

76.15.107 FileFlush

Synopsis: Synchronize file contents to disk.

Declaration: `function FileFlush(Handle: THandle) : Boolean`

Visibility: default

Description: `FileFlush` can be used to force the operating system to write any cached changes to file `Handle` to disk. It returns `True` if the operation completed succesfully. Note that not all operating systems support this operation.

Errors: On error, check the GetLastOSError (1751) function result for more information about the error.

See also: FileClose (1715), FileWrite (1726), FileOpen (1721)

76.15.108 FileGetAttr

Synopsis: Return attributes of a file.

Declaration: `function FileGetAttr(const FileName: UnicodeString) : LongInt`
`function FileGetAttr(const FileName: RawByteString) : LongInt`

Visibility: default

Description: `FileGetAttr` returns the attribute settings of file `FileName`. The attribute is a `OR`-ed combination of the following constants:

faReadOnly The file is read-only.

faHidden The file is hidden. (On UNIX, this means that the filename starts with a dot)

faSysFile The file is a system file (On UNIX, this means that the file is a character, block or FIFO file).

faVolumeId Volume Label. Only for DOS/Windows on a plain FAT (not VFAT or Fat32) file system.

faDirectory File is a directory.

faArchive file should be archived. Not possible on Unix

Errors: In case of error, -1 is returned.

See also: FileSetAttr (1724), FileAge (1714), FileGetDate (1719)

Listing: ./examples/sysutex/ex40.pp

```
Program Example40;

{ This program demonstrates the FileGetAttr function }

Uses sysutils;

Procedure Testit (Name : String);
```

```
Var F : Longint;

Begin
  F:=FileGetAttr(Name);
  If F<>-1 then
    begin
    WriteIn ('Testing : ',Name);
    If (F and faReadOnly)<>0 then
      WriteIn ('File is ReadOnly');
    If (F and faHidden)<>0 then
      WriteIn ('File is hidden');
    If (F and faSysFile)<>0 then
      WriteIn ('File is a system file');
    If (F and faVolumeID)<>0 then
      WriteIn ('File is a disk label');
    If (F and faArchive)<>0 then
      WriteIn ('File is artchive file');
    If (F and faDirectory)<>0 then
      WriteIn ('File is a directory');
    end
  else
   WriteIn ('Error reading attributes of ',Name);
end;

begin
  testit ('ex40.pp');
  testit (ParamStr(0));
  testit ('.');
  testit ('/');
End.
```

76.15.109 FileGetDate

Synopsis: Return the file time of an opened file.

Declaration:
```
function FileGetDate(Handle: THandle) : Int64
function FileGetDate(Handle: THandle; out FileDateTime: TDateTime)
                     : Boolean
```

Visibility: default

Description: `FileGetdate` returns the filetime of the opened file with file handle `Handle`. It is the same as FileAge (1714), with this difference that `FileAge` only needs the file name, while `FilegetDate` needs an open file handle.

Errors: On error, -1 is returned.

See also: FileAge (1714)

Listing: ./examples/sysutex/ex39.pp

```
Program Example39;

{ This program demonstrates the FileGetDate function }

Uses sysutils;
```

```
Var F,D : Longint;

Begin
  F:=FileCreate('test.dat');
  D:=FileGetDate(F);
  Writeln ('File created on ',DateTimeToStr(FileDateToDateTime(D)));
  FileClose(F);
  DeleteFile('test.dat');
End.
```

76.15.110 FileGetDateTimeInfo

Declaration:
```
function FileGetDateTimeInfo(const FileName: string;
                             out DateTime: TDateTimeInfoRec;
                             FollowLink: Boolean) : Boolean
```

Visibility: default

76.15.111 FileGetDateUTC

Declaration:
```
function FileGetDateUTC(Handle: THandle; out FileDateTimeUTC: TDateTime)
                        : Boolean
```

Visibility: default

76.15.112 FileGetSymLinkTarget

Synopsis: Get the target filename and size of a symlink.

Declaration:
```
function FileGetSymLinkTarget(const FileName: UnicodeString;
                              out SymLinkRec: TUnicodeSymLinkRec)
                               : Boolean
function FileGetSymLinkTarget(const FileName: UnicodeString;
                              out TargetName: UnicodeString) : Boolean
function FileGetSymLinkTarget(const FileName: RawByteString;
                              out SymLinkRec: TRawbyteSymLinkRec)
                               : Boolean
function FileGetSymLinkTarget(const FileName: RawByteString;
                              out TargetName: RawByteString) : Boolean
```

Visibility: default

Description: `FileGetSymLinkTarget` returns the target filename, size and attributes of a symbolic link in `SymLinkRec`. On unix-like systems, the filemode is also returned. The filename can be a relative filename, it will not be expanded.

See also: TUnicodeSymLinkRec (1828), TRawbyteSymLinkRec (1826)

76.15.113 FileIsReadOnly

Synopsis: Check whether a file is read-only.

Declaration:
```
function FileIsReadOnly(const FileName: UnicodeString) : Boolean
function FileIsReadOnly(const FileName: RawByteString) : Boolean
```

Visibility: default

Description: `FileIsReadOnly` checks whether `FileName` exists in the file system and is a read-only file. If this is the case, the function returns `True`, otherwise `False` is returned.

See also: FileExists (1717)

76.15.114 FileOpen

Synopsis: Open an existing file and return a file handle.

Declaration:
```
function FileOpen(const FileName: UnicodeString; Mode: Integer)
                  : THandle
function FileOpen(const FileName: RawByteString; Mode: Integer)
                  : THandle
```

Visibility: default

Description: `FileOpen` opens a file with name `FileName` with mode `Mode`. `Mode` can be one of the following constants:

fmOpenReadOpen file in read-only mode.

fmOpenWriteOpen file in write-only mode.

fmOpenReadWriteOpen file in read/write mode.

Under Windows and Unix, the above mode can be or-ed with one of the following sharing/locking flags:

fmShareCompatOpen file in DOS share-compatibility mode.

fmShareExclusiveLock file for exclusive use.

fmShareDenyWriteLock file so other processes can only read.

fmShareDenyReadLock file so other processes cannot read.

fmShareDenyNoneDo not lock file.

If the file has been successfully opened, it can be read from or written to (depending on the `Mode` parameter) with the FileRead (1722) and `FileWrite` functions.

Remark Remark that you cannot open a file if it doesn't exist yet, i.e. it will not be created for you. If you want tp create a new file, or overwrite an old one, use the FileCreate (1715) function.

There are some limitations to the sharing modes.

1. Sharing modes are only available on Unix and Windows platforms.
2. Unix only support sharing modes as of 2.4.0.
3. fmShareDenyRead only works under Windows at this time, and will always result in an error on Unix platforms because its file locking APIs do not support this concept.
4. File locking is advisory on Unix platforms. This means that the locks are only checked when a file is opened using a file locking mode. In other cases, existing locks are simply ignored. In particular, this means that `fmShareDenyNone` has no effect under Unix, because this can only be implemented as "use no locking" on those platforms. As a result, opening a file using this mode will always succeed under Unix as far as the locking is concerned, even if the file has already been opened using `fmShareExclusive`.

5. Under Solaris, closing a single file handle associated with a file will result in all locks on that file (even via other handles) being destroyed due to the behaviour of the underlying API (fcntl). Because of the same reason, on Solaris you cannot use `fmShareDenyWrite` in combination with `fmOpenWrite`, nor `fmShareExclusive` in combination with `fmOpenRead` although both work with fmOpenReadWrite.

For an example, see FileCreate (1715)

Errors: On Error, `THandle(-1)` is returned.

See also: fmOpenRead (1645), fmOpenWrite (1645), fmOpenReadWrite (1645), fmShareDenyWrite (1646), fmShareExclusive (1646), fmShareDenyRead (1646), fmShareDenyNone (1645), fmShareCompat (1645), FileClose (1715), FileWrite (1726), FileCreate (1715), FileRead (1722), FileTruncate (1725), FileSeek (1723)

76.15.115 FileRead

Synopsis: Read data from a file handle in a buffer.

Declaration:
```
function FileRead(Handle: THandle; out Buffer; Count: LongInt) : LongInt
```

Visibility: default

Description: `FileRead` reads `Count` bytes from file-handle `Handle` and stores them into `Buffer`. Buffer must be at least `Count` bytes long. No checking on this is performed, so be careful not to overwrite any memory. `Handle` must be the result of a FileOpen (1721) call.

The function returns the number of bytes actually read, or -1 on error.

For an example, see FileCreate (1715)

Errors: On error, -1 is returned.

See also: FileClose (1715), FileWrite (1726), FileCreate (1715), FileOpen (1721), FileTruncate (1725), FileSeek (1723)

76.15.116 FileSearch

Synopsis: Search for a file in a path.

Declaration:
```
function FileSearch(const Name: UnicodeString;
                    const DirList: UnicodeString;
                    Options: TFileSearchOptions) : UnicodeString
function FileSearch(const Name: UnicodeString;
                    const DirList: UnicodeString;
                    ImplicitCurrentDir: Boolean) : UnicodeString
function FileSearch(const Name: RawByteString;
                    const DirList: RawByteString;
                    Options: TFileSearchOptions) : RawByteString
function FileSearch(const Name: RawByteString;
                    const DirList: RawByteString;
                    ImplicitCurrentDir: Boolean) : RawByteString
```

Visibility: default

Description: `FileSearch` looks for the file `Name` in `DirList`, where dirlist is a list of directories, separated by semicolons or colons. It returns the full filename of the first match found. The optional `Options` parameter may be specified to influence the behaviour of the search algorithm. It is a set of the following options:

sfoImplicitCurrentDirAlways search the current directory first, even if it is not specified.

sfoStripQuotesStrip quotes from the components in the search path.

A deprecated form of the function allowed to specify using the boolean `ImplicitCurrentDir` parameter whether the current directory was searched implicitly or not. By default, the current directory is searched.

Errors: On error, an empty string is returned.

See also: ExpandFileName (1708), FindFirst (1727)

Listing: ./examples/sysutex/ex41.pp

```
Program Example41;

{ Program to demonstrate the FileSearch function. }

Uses Sysutils;

Const
{$ifdef unix}
  FN = 'find';
  P = '.:/bin:/usr/bin';
{$else}
  FN = 'find.exe';
  P = 'c:\dos;c:\windows;c:\windows\system;c:\windows\system32';
{$endif}

begin
  Writeln ('find is in : ',FileSearch (FN,P));
end.
```

76.15.117 FileSeek

Synopsis: Set the current file position on a file handle.

Declaration:
```
function FileSeek(Handle: THandle; FOffset: LongInt; Origin: LongInt)
                  : LongInt
function FileSeek(Handle: THandle; FOffset: Int64; Origin: LongInt)
                  : Int64
```

Visibility: default

Description: `FileSeek` sets the file pointer on position `Offset`, starting from `Origin`. Origin can be one of the following values:

fsFromBeginning`Offset` is relative to the first byte of the file. This position is zero-based. i.e. the first byte is at offset 0.

fsFromCurrent`Offset` is relative to the current position.

fsFromEnd`Offset` is relative to the end of the file. This means that `Offset` can only be zero or negative in this case.

If successful, the function returns the new file position, relative to the beginning of the file.

Remark The above mentioned constants do not exist in Delphi.

Errors: On error, -1 is returned.

See also: FileClose (1715), FileWrite (1726), FileCreate (1715), FileOpen (1721), FileRead (1722), FileTruncate (1725)

Listing: ./examples/sysutex/ex37.pp

```
Program Example37;

{ This program demonstrates the FileCreate/FileSeek/FileReed/FileTruncate functions }

Uses sysutils;

Var I,J,F : Longint;

Begin
  F:=FileCreate ('test.dat');
  If F=-1 then
    Halt(1);
  For I:=0 to 100 do
    FileWrite(F,I,SizeOf(i));
  FileClose(f);
  F:=FileOpen ('test.dat',fmOpenRead);
  For I:=0 to 100 do
    begin
    FileRead (F,J,SizeOF(J));
    If J<>I then
      WriteIn ('Mismatch at file position ',I)
    end;
  FileSeek(F,0,fsFromBeginning);
  Randomize;
  Repeat
    FileSeek(F,Random(100)*4,fsFromBeginning);
    FileRead (F,J,SizeOf(J));
    WriteIn ('Random read : ',j);
  Until J>80;
  FileClose(F);
  F:=FileOpen('test.dat',fmOpenWrite);
  I:=50*SizeOf(Longint);
  If FileTruncate(F,I) then
    WriteIn('SuccessFully truncated file to ',I,' bytes.');
  FileClose(F);
End.
```

76.15.118 FileSetAttr

Synopsis: Set the attributes of a file.

Declaration:
```
function FileSetAttr(const Filename: UnicodeString; Attr: LongInt)
                     : LongInt
function FileSetAttr(const Filename: RawByteString; Attr: LongInt)
                     : LongInt
```

Visibility: default

Description: `FileSetAttr` sets the attributes of `FileName` to `Attr`. If the function was successful, 0 is returned, -1 otherwise. `Attr` can be set to an OR-ed combination of the predefined `faXXX` constants.

This function is not implemented on Unixes.

Errors: On error, -1 is returned (always on Unixes).

See also: FileGetAttr (1718), FileGetDate (1719), FileSetDate (1725)

76.15.119 FileSetDate

Synopsis: Set the date of a file.

Declaration:
```
function FileSetDate(const FileName: UnicodeString; Age: Int64)
                         : LongInt
function FileSetDate(const FileName: UnicodeString;
                     const FileDateTime: TDateTime) : LongInt
function FileSetDate(const FileName: RawByteString; Age: Int64)
                         : LongInt
function FileSetDate(const FileName: RawByteString;
                     const FileDateTime: TDateTime) : LongInt
function FileSetDate(Handle: THandle; Age: Int64) : LongInt
function FileSetDate(Handle: THandle; const FileDateTime: TDateTime)
                         : LongInt
```

Visibility: default

Description: `FileSetDate` sets the file date of the open file with handle `Handle` or to `Age`, where `Age` is a DOS date-and-time stamp value.

Alternatively, the filename may be specified with the `FileName` argument. This variant of the call is mandatory on Unixes, since there is no OS support for setting a file timestamp based on a handle. (the handle may not be a real file at all).

The function returns zero if successful.

Errors: On Unix, the handle variant always returns -1, since this is impossible to implement. On Windows and DOS, a negative error code is returned.

76.15.120 FileSetDateUTC

Declaration:
```
function FileSetDateUTC(const FileName: UnicodeString;
                        const FileDateTimeUTC: TDateTime) : LongInt
function FileSetDateUTC(const FileName: RawByteString;
                        const FileDateTimeUTC: TDateTime) : LongInt
function FileSetDateUTC(Handle: THandle;
                        const FileDateTimeUTC: TDateTime) : LongInt
```

Visibility: default

76.15.121 FileTruncate

Synopsis: Truncate an open file to a given size.

Declaration:
```
function FileTruncate(Handle: THandle; Size: Int64) : Boolean
```

Visibility: default

Description: `FileTruncate` truncates the file with handle `Handle` to `Size` bytes. The file must have been opened for writing prior to this call. The function returns `True` is successful, `False` otherwise.

For an example, see FileCreate (1715).

Errors: On error, the function returns `False`.

See also: FileClose (1715), FileWrite (1726), FileCreate (1715), FileOpen (1721), FileRead (1722), FileSeek (1723)

76.15.122 FileWrite

Synopsis: Write data from a buffer to a given file handle.

Declaration:
```
function FileWrite(Handle: THandle; const Buffer; Count: LongInt)
                         : LongInt
```

Visibility: default

Description: `FileWrite` writes `Count` bytes from `Buffer` to the file with handle `Handle`. Prior to this call, the file must have been opened for writing. `Buffer` must be at least `Count` bytes large, or a memory access error may occur.

The function returns the number of bytes written, or -1 in case of an error.

For an example, see FileCreate (1715).

Errors: In case of error, -1 is returned.

See also: FileClose (1715), FileCreate (1715), FileOpen (1721), FileRead (1722), FileTruncate (1725), FileSeek (1723)

76.15.123 FindClose

Synopsis: Close a find handle.

Declaration:
```
procedure FindClose(var F: TUnicodeSearchRec)
procedure FindClose(var F: TRawbyteSearchRec)
```

Visibility: default

Description: `FindClose` ends a series of FindFirst (1727)/FindNext (1728) calls, and frees any memory used by these calls. It is *absolutely* necessary to do this call, or huge memory losses may occur.

For an example, see FindFirst (1727).

Errors: None.

See also: FindFirst (1727), FindNext (1728)

76.15.124 FindCmdLineSwitch

Synopsis: Check whether a certain switch is present on the command-line.

Declaration:
```
function FindCmdLineSwitch(const Switch: string;
                           const Chars: TSysCharSet; IgnoreCase: Boolean)
                            : Boolean
function FindCmdLineSwitch(const Switch: string; IgnoreCase: Boolean)
                            : Boolean
function FindCmdLineSwitch(const Switch: string) : Boolean
```

Visibility: default

Description: `FindCmdLineSwitch` will check all command-line arguments for the presence of the option `Switch`. It will return `True` if it was found, `False` otherwise. Characters that appear in `Chars` (default is SwitchChars (1650)) are assumed to indicate an option (switch). If the parameter `IgnoreCase` is `True`, case will be ignored when looking for the switch. Default is to search case sensitive.

Errors: None.

See also: SwitchChars (1650)

76.15.125 FindFirst

Synopsis: Start a file search and return a findhandle.

Declaration:
```
function FindFirst(const Path: UnicodeString; Attr: LongInt;
                   out Rslt: TUnicodeSearchRec) : LongInt
function FindFirst(const Path: RawByteString; Attr: LongInt;
                   out Rslt: TRawbyteSearchRec) : LongInt
```

Visibility: default

Description: `FindFirst` looks for files that match the name (possibly with wildcards) in `Path` and extra attributes `Attr`. It then fills up the `Rslt` record with data gathered about the file. It returns 0 if a file matching the specified criteria is found, a nonzero value (-1 on Unix-like platforms) otherwise.

`Attr` is an or-ed combination of the following constants:

faAnyFileFind any file (this is a combination of the other flags except `faSymlink`).

faReadOnlyThe file is read-only.

faHiddenThe file is hidden. (On UNIX, this means that the filename starts with a dot)

faSysFileThe file is a system file (On unix, this means that the file is a character, block or FIFO file).

faVolumeIdDrive volume Label. Not possible under unix, and on Windows-like systems, this works only for plan FAT (not Fat32 of VFAT) file systems.

faDirectoryFile is a directory.

faArchivefile needs to be archived. Not possible on Unix

faSymlinkReport symlinks instead of following them

It is a common misconception that `Attr` specifies a set of attributes which must be matched in order for a file to be included in the list. This is not so: The value of `Attr` specifies *additional* attributes, this means that the returned files are either normal files or have an attribute which is present in `Attr`.

Specifically: specifying `faDirectory` as a value for `Attr` does not mean that only directories will be returned. Normal files *and* directories will be returned.

Since `faSymlink` is not included in `faAnyFile`, to find all possible files and symlinks you need to call `FindFirst` so:

```
if FindFirst('*', faAnyFile or faSymLink, info)=0 then
  // do your thing
```

The `Rslt` record can be fed to subsequent calls to `FindNext`, in order to find other files matching the specifications.

Remark A successful `FindFirst` call must *always* be followed by a FindClose (1726) call with the same `Rslt` record. Failure to do so will result in memory leaks. If the findfirst call failed (i.e. returned a nonzero handle) there is no need to call `FindClose`.

Errors: On error the function returns -1 on Unix-like platforms, a nonzero error code on Windows.

See also: FindClose (1726), FindNext (1728)

Listing: ./examples/sysutex/ex43.pp

```
Program Example43;

{ This program demonstrates the FindFirst function }

Uses SysUtils;

Var Info : TSearchRec;
    Count : Longint;

Begin
  Count:=0;
  If FindFirst ('*',faAnyFile,Info)=0 then
    begin
    Repeat
      Inc(Count);
      With Info do
        begin
        If (Attr and faDirectory) = faDirectory then
          Write('Dir : ');
        WriteIn (Name:40,Size:15);
        end;
    Until FindNext(info)<>0;
    FindClose(Info);
    end;
  WriteIn ('Finished search. Found ',Count,' matches');
End.
```

76.15.126 FindNext

Synopsis: Find the next entry in a findhandle.

Declaration:
```
function FindNext(var Rslt: TUnicodeSearchRec) : LongInt
function FindNext(var Rslt: TRawbyteSearchRec) : LongInt
```

Visibility: default

Description: `FindNext` finds a next occurrence of a search sequence initiated by `FindFirst`. If another record matching the criteria in Rslt is found, 0 is returned, a nonzero constant is returned otherwise.

Remark The last `FindNext` call must *always* be followed by a `FindClose` call with the same `Rslt` record. Failure to do so will result in memory loss.

For an example, see FindFirst (1727)

Errors: On error (no more file is found), a nonzero constant is returned.

See also: FindFirst (1727), FindClose (1726)

76.15.127 FloattoCurr

Synopsis: Convert a float to a Currency value.

Declaration: `function FloattoCurr(const Value: Extended) : Currency`

Visibility: default

Description: `FloatToCurr` converts the `Value` floating point value to a `Currency` value. It checks whether `Value` is in the valid range of currencies (determined by MinCurrency (1647) and MaxCurrency (1647)). If not, an EConvertError (1830) exception is raised.

Errors: If `Value` is out of range, an EConvertError (1830) exception is raised.

See also: EConvertError (1830), TryFloatToCurr (1805), MinCurrency (1647), MaxCurrency (1647)

76.15.128 FloatToDateTime

Synopsis: Convert a float to a `TDateTime` value.

Declaration: `function FloatToDateTime(const Value: Extended) : TDateTime`

Visibility: default

Description: `FloatToDateTime` converts the `Value` floating point value to a `TDateTime` value. It checks whether `Value` is in the valid range of dates (determined by MinDateTime (1647) and MaxDateTime (1647)). If not, an EConvertError (1830) exception is raised.

Errors: If `Value` is out of range, an EConvertError (1830) exception is raised.

See also: EConvertError (1830), MinDateTime (1647), MaxDateTime (1647)

76.15.129 FloatToDecimal

Synopsis: Convert a float value to a `TFloatRec` value.

Declaration:
```
procedure FloatToDecimal(out Result: TFloatRec; const Value;
                         ValueType: TFloatValue; Precision: Integer;
                         Decimals: Integer)
procedure FloatToDecimal(out Result: TFloatRec; Value: Extended;
                         Precision: Integer; Decimals: Integer)
```

Visibility: default

Description: `FloatToDecimal` converts the float `Value` to a float description in the `Result`TFloatRec (1656) format. It will store `Precision` digits in the `Digits` field, of which at most `Decimal` decimals.

Errors: None.

See also: TFloatRec (1656)

76.15.130 FloatToStr

Synopsis: Convert a float value to a string using a fixed format.

Declaration:
```
function FloatToStr(Value: Extended) : string
function FloatToStr(Value: Extended;
                    const FormatSettings: TFormatSettings) : string
function FloatToStr(Value: Double) : string
function FloatToStr(Value: Double;
```

```
                    const FormatSettings: TFormatSettings) : string
function FloatToStr(Value: Single) : string
function FloatToStr(Value: Single;
                    const FormatSettings: TFormatSettings) : string
function FloatToStr(Value: Currency) : string
function FloatToStr(Value: Currency;
                    const FormatSettings: TFormatSettings) : string
function FloatToStr(Value: Comp) : string
function FloatToStr(Value: Comp; const FormatSettings: TFormatSettings)
                     : string
function FloatToStr(Value: Int64) : string
function FloatToStr(Value: Int64; const FormatSettings: TFormatSettings)
                    : string
```

Visibility: default

Description: `FloatToStr` converts the floating point variable `Value` to a string representation. It will choose the shortest possible notation of the two following formats:

Fixed formatwill represent the string in fixed notation,
Decimal formatwill represent the string in scientific notation.

More information on these formats can be found in FloatToStrF (1731). `FloatToStr` is completely equivalent to the following call:

```
FloatToStrF(Value, ffGeneral,15, 0);
```

Note that on unix systems, the localization support must be enabled explicitly, see Localization (1636).

Errors: None.

See also: FloatToStrF (1731), FormatFloat (1743), StrToFloat (1791)

Listing: ./examples/sysutex/ex67.pp

```
Program Example67;

{ This program demonstrates the FloatToStr function }

Uses sysutils;

Procedure Testit (Value : Extended);

begin
  WriteLn (Value,' -> ',FloatToStr(Value));
  WriteLn (-Value,' -> ',FloatToStr(-Value));
end;

Begin
  Testit (0.0);
  Testit (1.1);
  Testit (1.1e-3);
  Testit (1.1e-20);
  Testit (1.1e-200);
  Testit (1.1e+3);
  Testit (1.1e+20);
  Testit (1.1e+200);
End.
```

76.15.131 FloatToStrF

Synopsis: Convert a float value to a string using a given format.

Declaration:
```
function FloatToStrF(Value: Extended; format: TFloatFormat;
                     Precision: Integer; Digits: Integer) : string
function FloatToStrF(Value: Extended; format: TFloatFormat;
                     Precision: Integer; Digits: Integer;
                     const FormatSettings: TFormatSettings) : string
function FloatToStrF(Value: Double; format: TFloatFormat;
                     Precision: Integer; Digits: Integer) : string
function FloatToStrF(Value: Double; format: TFloatFormat;
                     Precision: Integer; Digits: Integer;
                     const FormatSettings: TFormatSettings) : string
function FloatToStrF(Value: Single; format: TFloatFormat;
                     Precision: Integer; Digits: Integer) : string
function FloatToStrF(Value: Single; format: TFloatFormat;
                     Precision: Integer; Digits: Integer;
                     const FormatSettings: TFormatSettings) : string
function FloatToStrF(Value: Comp; format: TFloatFormat;
                     Precision: Integer; Digits: Integer) : string
function FloatToStrF(Value: Comp; format: TFloatFormat;
                     Precision: Integer; Digits: Integer;
                     const FormatSettings: TFormatSettings) : string
function FloatToStrF(Value: Currency; format: TFloatFormat;
                     Precision: Integer; Digits: Integer) : string
function FloatToStrF(Value: Currency; format: TFloatFormat;
                     Precision: Integer; Digits: Integer;
                     const FormatSettings: TFormatSettings) : string
function FloatToStrF(Value: Int64; format: TFloatFormat;
                     Precision: Integer; Digits: Integer) : string
function FloatToStrF(Value: Int64; format: TFloatFormat;
                     Precision: Integer; Digits: Integer;
                     const FormatSettings: TFormatSettings) : string
```

Visibility: default

Description: `FloatToStrF` converts the floating point number `value` to a string representation, according to the settings of the parameters `Format`, `Precision` and `Digits`.

The meaning of the `Precision` and `Digits` parameter depends on the `Format` parameter. The format is controlled mainly by the `Format` parameter. It can have one of the following values:

ffcurrency Money format. `Value` is converted to a string using the global variables `CurrencyString`, `CurrencyFormat` and `NegCurrFormat`. The `Digits` parameter specifies the number of digits following the decimal point and should be in the range -1 to 18. If Digits equals `-1`, `CurrencyDecimals` is assumed. The `Precision` parameter is ignored.

ffExponent Scientific format. `Value` is converted to a string using scientific notation: 1 digit before the decimal point, possibly preceded by a minus sign if `Value` is negative. The number of digits after the decimal point is controlled by `Precision` and must lie in the range 0 to 15.

ffFixed Fixed point format. `Value` is converted to a string using fixed point notation. The result is composed of all digits of the integer part of `Value`, preceded by a minus sign if `Value` is negative. Following the integer part is `DecimalSeparator` and then the fractional part of `Value`, rounded off to `Digits` numbers. If the number is too large then the result will be in scientific notation.

ffGeneralGeneral number format. The argument is converted to a string using `ffExponent` or `ffFixed` format, depending on which one gives the shortest string. There will be no trailing zeroes. If `Value` is less than `0.00001` or if the number of decimals left of the decimal point is larger than `Precision` then scientific notation is used, and `Digits` is the minimum number of digits in the exponent. Otherwise `Digits` is ignored.

ffnumberIs the same as `ffFixed`, except that thousand separators are inserted in the resulting string.

Errors: None.

See also: FloatToStr (1729), FloatToText (1732)

Listing: ./examples/sysutex/ex68.pp

```
Program Example68;

{ This program demonstrates the FloatToStrF function }

Uses sysutils;

Const Fmt : Array [TFloatFormat] of string[10] =
          ('general','exponent','fixed','number','Currency');

Procedure Testit (Value :  Extended);

Var I,J : longint;
    FF : TFloatFormat;

begin
  For I:=5 to 15 do
    For J:=1 to 4 do
      For FF:=ffgeneral to ffcurrency do
        begin
        Write (Value,'(Prec: ',I:2,', Dig: ',J,', fmt : ',Fmt[ff],') : ');
        WriteIn (FloatToStrf(Value,FF,I,J));
        Write (-Value,'(Prec: ',I:2,', Dig: ',J,', fmt : ',Fmt[ff],') : ');
        WriteIn (FloatToStrf(-Value,FF,I,J));
        end;
end;

Begin
  Testit (1.1);
  Testit (1.1E1);
  Testit (1.1E-1);
  Testit (1.1E5);
  Testit (1.1E-5);
  Testit (1.1E10);
  Testit (1.1E-10);
  Testit (1.1E15);
  Testit (1.1E-15);
  Testit (1.1E100);
  Testit (1.1E-100);
End.
```

76.15.132 FloatToText

Synopsis: Return a string representation of a float, with a given format.

Declaration:
```
function FloatToText(Buffer: PChar; Value: Extended;
                     format: TFloatFormat; Precision: Integer;
                     Digits: Integer) : LongInt
function FloatToText(Buffer: PChar; Value: Extended;
                     format: TFloatFormat; Precision: Integer;
                     Digits: Integer;
                     const FormatSettings: TFormatSettings) : LongInt
```

Visibility: default

Description: `FloatToText` converts the floating point variable `Value` to a string representation and stores it in `Buffer`. The conversion is governed by `format`, `Precision` and `Digits`. more information on these parameters can be found in FloatToStrF (1731). `Buffer` should point to enough space to hold the result. No checking on this is performed.

The result is the number of characters that was copied in `Buffer`.

Errors: None.

See also: FloatToStr (1729), FloatToStrF (1731)

Listing: ./examples/sysutex/ex69.pp

```
Program Example68;

{ This program demonstrates the FloatToStrF function }

Uses sysutils;

Const Fmt : Array [TFloatFormat] of string[10] =
        ('general','exponent','fixed','number','Currency');

Procedure Testit (Value :  Extended);

Var I,J : longint;
    FF : TFloatFormat;
    S : ShortString;

begin
  For I:=5 to 15 do
    For J:=1 to 4 do
      For FF:=ffgeneral to ffcurrency do
        begin
        Write (Value,'(Prec: ',I:2,', Dig: ',J,', fmt : ',Fmt[ff],') : ');
        SetLength(S,FloatToText (@S[1],Value,FF,I,J));
        WriteIn (S);
        Write (-Value,'(Prec: ',I:2,', Dig: ',J,', fmt : ',Fmt[ff],') : ');
        SetLength(S,FloatToText (@S[1],-Value,FF,I,J));
        WriteIn (S);
        end;
end;

Begin
  Testit (1.1);
  Testit (1.1E1);
  Testit (1.1E-1);
  Testit (1.1E5);
  Testit (1.1E-5);
  Testit (1.1E10);
```

```
  Testit (1.1E-10);
  Testit (1.1E15);
  Testit (1.1E-15);
  Testit (1.1E100);
  Testit (1.1E-100);
End.
```

76.15.133 FloatToTextFmt

Synopsis: Convert a float value to a string using a given mask.

Declaration:
```
function FloatToTextFmt(Buffer: PChar; Value: Extended; format: PChar;
                        FormatSettings: TFormatSettings) : Integer
function FloatToTextFmt(Buffer: PChar; Value: Extended; format: PChar)
                        : Integer
```

Visibility: default

Description: `FloatToTextFmt` returns a textual representation of `Value` in the memory location pointed to by `Buffer`. it uses the formatting specification in `Format` to do this. The return value is the number of characters that were written in the buffer.

For a list of valid formatting characters, see FormatFloat (1743)

Errors: No length checking is performed on the buffer. The buffer should point to enough memory to hold the complete string. If this is not the case, an access violation may occur.

See also: FormatFloat (1743)

76.15.134 FmtStr

Synopsis: Format a string with given arguments.

Declaration:
```
procedure FmtStr(var Res: string; const Fmt: string;
                 const args: Array of const)
procedure FmtStr(var Res: string; const Fmt: string;
                 const args: Array of const;
                 const FormatSettings: TFormatSettings)
```

Visibility: default

Description: `FmtStr` calls Format (1735) with `Fmt` and `Args` as arguments, and stores the result in `Res`. For more information on how the resulting string is composed, see Format (1735).

Errors: In case of error, a EConvertError (1830) exception is raised.

See also: Format (1735), FormatBuf (1742), EConvertError (1830)

Listing: ./examples/sysutex/ex70.pp

```
Program Example70;

{ This program demonstrates the FmtStr function }

Uses sysutils;

Var S : AnsiString;
```

```
Begin
  S:='';
  FmtStr (S,'For some nice examples of fomatting see %s.',['Format']);
  Writeln (S);
End.
```

76.15.135 ForceDirectories

Synopsis: Create a chain of directories.

Declaration:
```
function ForceDirectories(const Dir: RawByteString) : Boolean
function ForceDirectories(const Dir: UnicodeString) : Boolean
```

Visibility: default

Description: `ForceDirectories` tries to create any missing directories in `Dir` till the whole path in `Dir` exists. It returns `True` if `Dir` already existed or was created successfully. If it failed to create any of the parts, `False` is returned.

76.15.136 Format

Synopsis: Format a string with given arguments.

Declaration:
```
function Format(const Fmt: string; const Args: Array of const) : string
function Format(const Fmt: string; const Args: Array of const;
                const FormatSettings: TFormatSettings) : string
```

Visibility: default

Description: Format replaces all placeholders in`Fmt` with the arguments passed in `Args` and returns the resulting string. A placeholder looks as follows:

```
'%' [[Index]':'] ['-'] [Width] ['.' Precision] ArgType
```

elements between single quotes must be typed as shown without the quotes, and elements between square brackets `[ ]` are optional. The meaning of the different elements are shown below:

'%'starts the placeholder. If you want to insert a literal `%` character, then you must insert two of them : `%%`.

Index ':'takes the `Index`-th element in the argument array as the element to insert. If `index` is omitted, then the zeroth argument is taken.

'-'tells `Format` to left-align the inserted text. The default behaviour is to right-align inserted text. This can only take effect if the `Width` element is also specified.

Widththe inserted string must have at least `Width` characters. If not, the inserted string will be padded with spaces. By default, the string is left-padded, resulting in a right-aligned string. This behaviour can be changed by the usage of the `'-'` character.

'.' Precision Indicates the precision to be used when converting the argument. The exact meaning of this parameter depends on `ArgType`.

The `Index`, `Width` and `Precision` parameters can be replaced by `*`, in which case their value will be read from the next element in the `Args` array. This value must be an integer, or an EConvertError (1830) exception will be raised.

The argument type is determined from `ArgType`. It can have one of the following values (case insensitive):

BBoolean format. The next argument in the `Args` array should be a boolean or an integer (a nonzero value is taken as `True`, zero is taken as false). The argument is converted to a string using BoolToStr (1685) with `True` as the second argument, i.e. boolean strings are used. If a precision is specified, then the string will have at most `Precision` characters in it, as for the "S" format specifier.

DDecimal format. The next argument in the `Args` array should be an integer. The argument is converted to a decimal string. If precision is specified, then the string will have at least `Precision` digits in it. If needed, the string is (left) padded with zeroes.

EScientific format. The next argument in the `Args` array should be a Floating point value. The argument is converted to a decimal string using scientific notation, using FloatToStrF (1731), where the optional precision is used to specify the total number of decimals. (default a value of 15 is used). The exponent is formatted using maximally 3 digits.

In short, the `E` specifier formats it's argument as follows:

```
FloatToStrF(Argument,ffexponent,Precision,3)
```

FFixed point format. The next argument in the `Args` array should be a floating point value. The argument is converted to a decimal string, using fixed notation (see FloatToStrF (1731)). `Precision` indicates the number of digits following the decimal point.

In short, the `F` specifier formats its argument as follows:

```
FloatToStrF(Argument,ffFixed,ffixed,9999,Precision)
```

GGeneral number format. The next argument in the `Args` array should be a floating point value. The argument is converted to a decimal string using fixed point notation or scientific notation, depending on which gives the shortest result. `Precision` is used to determine the number of digits after the decimal point.

In short, the `G` specifier formats it's argument as follows:

```
FloatToStrF(Argument,ffGeneral,Precision,3)
```

MCurrency format. the next argument in the `Args` array must be a floating point value. The argument is converted to a decimal string using currency notation. This means that fixed-point notation is used, but that the currency symbol is appended. If precision is specified, then then it overrides the `CurrencyDecimals` global variable used in the FloatToStrF (1731)

In short, the `M` specifier formats it's argument as follows:

```
FloatToStrF(Argument,ffCurrency,9999,Precision)
```

NNumber format. This is the same as fixed point format, except that thousand separators are inserted in the resulting string.

PPointer format. The next argument in the `Args` array must be a pointer (typed or untyped). The pointer value is converted to a string of length 8, representing the hexadecimal value of the pointer.

SString format. The next argument in the `Args` array must be a string or a character (unicode or ansi is acceptable for both). The argument is simply copied to the result string. If `Precision` is specified, then only `Precision` characters are copied to the result string.

UUnsigned decimal format. The next argument in the `Args` array should be an unsigned integer. The argument is converted to a decimal string. If precision is specified, then the string will have at least `Precision` digits in it. If needed, the string is (left) padded with zeroes.

Xhexadecimal format. The next argument in the `Args` array must be an integer. The argument is converted to a hexadecimal string with just enough characters to contain the value of the integer. If `Precision` is specified then the resulting hexadecimal representation will have at least `Precision` characters in it (with a maximum value of 32).

An unknown formatting character is ignored.

Errors: In case of error, an `EConvertError` exception is raised. Possible errors are:

1. Errors in the format specifiers.
2. The next argument is not of the type needed by a specifier.
3. The number of arguments is not sufficient for all format specifiers.

See also: FormatBuf (1742), EConvertError (1830)

Listing: ./examples/sysutex/ex71.pp

```
Program example71;

{$mode objfpc}

{ This program demonstrates the Format function }

Uses sysutils;

Var P : Pointer;
    fmt,S : string;

{ Expected output:
        [%d] => [10]
        [%%] => [%]
      [%10d] => [        10]
       [%.4d] => [0010]
    [%10.4d] => [      0010]
      [%0:d] => [10]
    [%0:10d] => [        10]
  [%0:10.4d] => [      0010]
   [%0:-10d] => [10        ]
 [%0:-10.4d] => [0010      ]
    [%-*.*d] => [00010]
}
Procedure TestInteger;
begin
  Try
    Fmt:='[%d]';S:=Format(Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%%]';S:=Format(Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%10d]';S:=Format(Fmt,[10]);writeln(Fmt:12,' => ',s);
    fmt:='[%.4d]';S:=Format(fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%10.4d]';S:=Format(Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:d]';S:=Format(Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:10d]';S:=Format(Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:10.4d]';S:=Format(Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:-10d]';S:=Format(Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:-10.4d]';S:=Format(fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%-*.*d]';S:=Format(fmt,[4,5,10]);writeln(Fmt:12,' => ',s);
  except
    On E : Exception do
      begin
      Writeln('Exception caught : ',E.Message);
      end;
  end;
  writeln('Press enter');
  readln;
```

```
end;

{ Expected output:
        [%x] => [A]
       [%10x] => [         A]
     [%10.4x] => [      000A]
       [%0:x] => [A]
     [%0:10x] => [         A]
   [%0:10.4x] => [      000A]
    [%0:-10x] => [A         ]
  [%0:-10.4x] => [000A      ]
     [%-*.*x] => [0000A]
}
Procedure TestHexaDecimal;
begin
  try
    Fmt:='[%x]';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%10x]';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%10.4x]';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:x]';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:10x]';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:10.4x]';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:-10x]';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:-10.4x]';S:=Format (fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='[%-*.*x]';S:=Format (fmt,[4,5,10]);writeln(Fmt:12,' => ',s);
  except
    On E : Exception do
      begin
      Writeln ('Exception caught : ',E.Message);
      end;
  end;
  writeln ('Press enter');
  readln;
end;

{ Expected output:
        [0x%p] => [0x0012D687]
      [0x%10p] => [0x  0012D687]
    [0x%10.4p] => [0x  0012D687]
      [0x%0:p] => [0x0012D687]
    [0x%0:10p] => [0x  0012D687]
  [0x%0:10.4p] => [0x  0012D687]
   [0x%0:-10p] => [0x0012D687  ]
 [0x%0:-10.4p] => [0x0012D687  ]
     [%-*.*p] => [0012D687]
}
Procedure TestPointer;
begin
  P:=Pointer(1234567);
  try
    Fmt:='[0x%p]';S:=Format (Fmt,[P]);writeln(Fmt:12,' => ',s);
    Fmt:='[0x%10p]';S:=Format (Fmt,[P]);writeln(Fmt:12,' => ',s);
    Fmt:='[0x%10.4p]';S:=Format (Fmt,[P]);writeln(Fmt:12,' => ',s);
    Fmt:='[0x%0:p]';S:=Format (Fmt,[P]);writeln(Fmt:12,' => ',s);
    Fmt:='[0x%0:10p]';S:=Format (Fmt,[P]);writeln(Fmt:12,' => ',s);
    Fmt:='[0x%0:10.4p]';S:=Format (Fmt,[P]);writeln(Fmt:12,' => ',s);
    Fmt:='[0x%0:-10p]';S:=Format (Fmt,[P]);writeln(Fmt:12,' => ',s);
    Fmt:='[0x%0:-10.4p]';S:=Format (fmt,[P]);writeln(Fmt:12,' => ',s);
```

```
    Fmt:='[%-*.*p]';S:=Format (fmt,[4,5,P]);writeln(Fmt:12,' => ',s);
  except
    On E : Exception do
      begin
      WriteIn ('Exception caught : ',E.Message);
      end;
  end;
  writeln ('Press enter');
  readln;
end;

{ Expected output:
        [%s]=> [This is a string]
      [%0:s]=> [This is a string]
    [%0:18s]=> [  This is a string]
   [%0:-18s]=> [This is a string  ]
 [%0:18.12s]=> [      This is a st]
    [%-*.*s]=> [This is a st      ]
}
Procedure TestString;
begin
  try
    Fmt:='[%s]';S:=Format(fmt,['This is a string']);Writeln(fmt:12,'=> ',s);
    fmt:='[%0:s]';s:=Format(fmt,['This is a string']);Writeln(fmt:12,'=> ',s);
    fmt:='[%0:18s]';s:=Format(fmt,['This is a string']);Writeln(fmt:12,'=> ',s);
    fmt:='[%0:-18s]';s:=Format(fmt,['This is a string']);Writeln(fmt:12,'=> ',s);
    fmt:='[%0:18.12s]';s:=Format(fmt,['This is a string']);Writeln(fmt:12,'=> ',s);
    fmt:='[%-*.*s]';s:=Format(fmt,[18,12,'This is a string']);Writeln(fmt:12,'=> ',s);
  except
    On E : Exception do
      begin
      WriteIn ('Exception caught : ',E.Message);
      end;
  end;
  writeln ('Press enter');
  readln;
end;

{ Expected output:
        [%e] => [1.2340000000000000E+000]
      [%10e] => [1.2340000000000000E+000]
    [%10.4e] => [1.234E+000]
      [%0:e] => [1.2340000000000000E+000]
    [%0:10e] => [1.2340000000000000E+000]
  [%0:10.4e] => [1.234E+000]
   [%0:-10e] => [1.2340000000000000E+000]
 [%0:-10.4e] => [1.234E+000]
    [%-*.*e] => [1.2340E+000]
}
Procedure TestExponential;
begin
  Try
    Fmt:='[%e]';S:=Format (Fmt,[1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%10e]';S:=Format (Fmt,[1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%10.4e]';S:=Format (Fmt,[1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:e]';S:=Format (Fmt,[1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:10e]';S:=Format (Fmt,[1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:10.4e]';S:=Format (Fmt,[1.234]);writeln(Fmt:12,' => ',s);
```

```
    Fmt:='[%0:-10e]';S:=Format (Fmt,[1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:-10.4e]';S:=Format (fmt,[1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%-*.*e]';S:=Format (fmt,[4,5,1.234]);writeln(Fmt:12,' => ',s);
  except
    On E : Exception do
      begin
      WriteIn ('Exception caught : ',E.Message);
      end;
  end;
  writeln ('Press enter');
  readln;
end;

{ Expected output:
        [%e] => [-1.2340000000000000E+000]
      [%10e] => [-1.2340000000000000E+000]
    [%10.4e] => [-1.234E+000]
      [%0:e] => [-1.2340000000000000E+000]
    [%0:10e] => [-1.2340000000000000E+000]
  [%0:10.4e] => [-1.234E+000]
   [%0:-10e] => [-1.2340000000000000E+000]
 [%0:-10.4e] => [-1.234E+000]
    [%-*.*e] => [-1.2340E+000]
}
Procedure TestNegativeExponential;
begin
  Try
    Fmt:='[%e]';S:=Format (Fmt,[-1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%10e]';S:=Format (Fmt,[-1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%10.4e]';S:=Format (Fmt,[-1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:e]';S:=Format (Fmt,[-1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:10e]';S:=Format (Fmt,[-1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:10.4e]';S:=Format (Fmt,[-1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:-10e]';S:=Format (Fmt,[-1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:-10.4e]';S:=Format (fmt,[-1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%-*.*e]';S:=Format (fmt,[4,5,-1.234]);writeln(Fmt:12,' => ',s);
  except
    On E : Exception do
      begin
      WriteIn ('Exception caught : ',E.Message);
      end;
  end;
  writeln ('Press enter');
  readln;
end;

{ Expected output:
        [%e] => [1.2340000000000000E-002]
      [%10e] => [1.2340000000000000E-002]
    [%10.4e] => [1.234E-002]
      [%0:e] => [1.2340000000000000E-002]
    [%0:10e] => [1.2340000000000000E-002]
  [%0:10.4e] => [1.234E-002]
   [%0:-10e] => [1.2300000000000000E-002]
 [%0:-10.4e] => [1.234E-002]
    [%-*.*e] => [1.2340E-002]
}
Procedure TestSmallExponential;
```

```
begin
  Try
    Fmt:='[%e]';S:=Format (Fmt,[0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%10e]';S:=Format (Fmt,[0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%10.4e]';S:=Format (Fmt,[0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:e]';S:=Format (Fmt,[0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:10e]';S:=Format (Fmt,[0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:10.4e]';S:=Format (Fmt,[0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:-10e]';S:=Format (Fmt,[0.0123]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:-10.4e]';S:=Format (fmt,[0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%-*.*e]';S:=Format (fmt,[4,5,0.01234]);writeln(Fmt:12,' => ',s);
  except
    On E : Exception do
      begin
      Writeln ('Exception caught : ',E.Message);
      end;
  end;
  writeln ('Press enter');
  readln;
end;

{ Expected output:
        [%e] => [-1.2340000000000000E-002]
      [%10e] => [-1.2340000000000000E-002]
    [%10.4e] => [-1.234E-002]
      [%0:e] => [-1.2340000000000000E-002]
    [%0:10e] => [-1.2340000000000000E-002]
  [%0:10.4e] => [-1.234E-002]
   [%0:-10e] => [-1.2340000000000000E-002]
 [%0:-10.4e] => [-1.234E-002]
    [%-*.*e] => [-1.2340E-002]
}
Procedure TestSmallNegExponential;
begin
  Try
    Fmt:='[%e]';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%10e]';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%10.4e]';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:e]';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:10e]';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:10.4e]';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:-10e]';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%0:-10.4e]';S:=Format (fmt,[-0.01234]);writeln(Fmt:12,' => ',s);
    Fmt:='[%-*.*e]';S:=Format (fmt,[4,5,-0.01234]);writeln(Fmt:12,' => ',s);
  except
    On E : Exception do
      begin
      Writeln ('Exception caught : ',E.Message);
      end;
  end;
  writeln ('Press enter');
  readln;
end;

begin
  TestInteger;
  TestHexadecimal;
  TestPointer;
```

```
  teststring;
  TestExponential;
  TestNegativeExponential;
  TestSmallExponential;
  TestSmallNegExponential;
end.
```

76.15.137 FormatBuf

Synopsis: Format a string with given arguments and store the result in a buffer.

Declaration:
```
function FormatBuf(var Buffer; BufLen: Cardinal; const Fmt;
                   fmtLen: Cardinal; const Args: Array of const)
                    : Cardinal
function FormatBuf(var Buffer; BufLen: Cardinal; const Fmt;
                   fmtLen: Cardinal; const Args: Array of const;
                   const FormatSettings: TFormatSettings) : Cardinal
```

Visibility: default

Description: `FormatBuf` calls Format (1735) and stores the result in `Buf`.

See also: Format (1735)

Listing: ./examples/sysutex/ex72.pp

```
Program Example72;

{ This program demonstrates the FormatBuf function }

Uses sysutils;

Var
  S : ShortString;

Const
  Fmt : ShortString =  'For some nice examples of fomatting see %s.';

Begin
  S:='';
  SetLength(S,FormatBuf (S[1],255,Fmt[1],Length(Fmt),['Format']));
  Writeln (S);
End.
```

76.15.138 FormatCurr

Synopsis: Format a currency.

Declaration:
```
function FormatCurr(const Format: string; Value: Currency) : string
function FormatCurr(const Format: string; Value: Currency;
                    const FormatSettings: TFormatSettings) : string
```

Visibility: default

Description: `FormatCurr` formats the currency `Value` according to the formatting rule in the `Format` parameter, and returns the resulting string.

For an explanation of the formatting characters usable in the `Format` parameter, see FormatFloat (1743).

See also: FormatFloat (1743), FloatToText (1732)

76.15.139 FormatDateTime

Synopsis: Return a string representation of a `TDateTime` value with a given format.

Declaration:
```
function FormatDateTime(const FormatStr: string; DateTime: TDateTime;
                        Options: TFormatDateTimeOptions) : string
function FormatDateTime(const FormatStr: string; DateTime: TDateTime;
                        const FormatSettings: TFormatSettings;
                        Options: TFormatDateTimeOptions) : string
```

Visibility: default

Description: `FormatDateTime` formats the date and time encoded in `DateTime` according to the formatting given in `FormatStr`. The complete list of formatting characters can be found in formatchars (1641).

When the format string is empty, 'c' is used instead.

Note that on unix systems, the localization support must be enabled explicitly, see Localization (1636).

Errors: On error (such as an invalid character in the formatting string), and EConvertError (1830) exception is raised.

See also: DateTimeToStr (1695), DateToStr (1698), TimeToStr (1802), StrToDateTime (1790), EConvertError (1830), Localization (1636)

Listing: ./examples/sysutex/ex14.pp

```
Program Example14;

{ This program demonstrates the FormatDateTime function }

Uses sysutils;

Var ThisMoment : TDateTime;

Begin
  ThisMoment:=Now;
  Writeln ('Now : ',FormatDateTime('hh:nn',ThisMoment));
  Writeln ('Now : ',FormatDateTime('DD MM YYYY',ThisMoment));
  Writeln ('Now : ',FormatDateTime('c',ThisMoment));
End.
```

76.15.140 FormatFloat

Synopsis: Format a float according to a certain mask.

Declaration:
```
function FormatFloat(const Format: string; Value: Extended) : string
function FormatFloat(const Format: string; Value: Extended;
                     const FormatSettings: TFormatSettings) : string
```

Visibility: default

Description: `FormatFloat` formats the floating-point value given by `Value` using the format specifications in `Format`. The format specifier can give format specifications for positive, negative or zero values (separated by a semicolon). Note that the minus sign must be explicitly specified in the section for negative numbers:

```
Writeln(FormatFloat('#0.00;#0.00-;0",FF));
```

If the format specifier is empty or the value needs more than 18 digits to be correctly represented, the result is formatted with a call to FloatToStrF (1731) with the `ffGeneral` format option.

The following format specifiers are supported:

0 is a digit place holder. If there is a corresponding digit in the value being formatted, then it replaces the 0. If not, the 0 is left as-is.

is also a digit place holder. If there is a corresponding digit in the value being formatted, then it replaces the #. If not, it is removed. by a space.

. determines the location of the decimal point. Only the first '.' character is taken into account. If the value contains digits after the decimal point, then it is replaced by the value of the `DecimalSeparator` character.

, determines the use of the thousand separator character in the output string. If the format string contains one or more ',' characters, then thousand separators will be used. The `ThousandSeparator` character is used.

E+ determines the use of scientific notation. If 'E+' or 'E-' (or their lowercase counterparts) are present then scientific notation is used. The number of digits in the output string is determined by the number of `0` characters after the '`E+`'

; This character separates sections for positive, negative, and zero numbers in the format string.

Errors: If an error occurs, an exception is raised.

See also: FloatToStr (1729)

Listing: ./examples/sysutex/ex89.pp

```
Program Example89;

{ This program demonstrates the FormatFloat function }

Uses sysutils;

Const
  NrFormat=9;
  FormatStrings : Array[1..NrFormat] of string = (
        '',
        '0',
        '0.00',
        '#.##',
        '#,##0.00',
        '#,##0.00;(#,##0.00)',
        '#,##0.00;;Zero',
        '0.000E+00',
        '#.###E-0');
  NrValue = 5;
  FormatValues : Array[1..NrValue] of Double =
    (1234,-1234,0.5,0,-0.5);
```

```
  Width  = 12;
  FWidth = 20;

Var
  I,J : Integer;
  S : String;

begin
  Write('Format':FWidth);
  For I:=1 to NrValue do
    Write(FormatValues[i]:Width:2);
  WriteIn;
  For I:=1 to NrFormat do
    begin
    Write(FormatStrings[i]:FWidth);
    For J:=1 to NrValue do
      begin
      S:=FormatFloat(FormatStrings[I],FormatValues[j]);
      Write(S:Width);
      end;
    WriteIn;
    end;
End.
```

76.15.141 FreeAndNil

Synopsis: Free object if needed, and set object reference to `Nil`.

Declaration: `procedure FreeAndNil(var obj)`

Visibility: default

Description: `FreeAndNil` will free the object in `Obj` and will set the reference in `Obj` to `Nil`. The reference is set to `Nil` first, so if an exception occurs in the destructor of the object, the reference will be `Nil` anyway.

Errors: Exceptions that occur during the destruction of `Obj` are not caught.

76.15.142 FreeMemAndNil

Synopsis: Free the heap memory pointed to by a pointer and set pointer to nil.

Declaration: `procedure FreeMemAndNil(var p)`

Visibility: default

Description: `FreeMemAndNil` will free the memory pointed to by `P` and will set `P` to Nil. Note that this function is not type safe; the programmer is responsible for ensuring that this function is called with a correct pointer parameter. Failure to do so may result in run-time errors.

See also: FreeAndNil (1745)

76.15.143 GetAppConfigDir

Synopsis: Return the appropriate directory for the application's configuration files.

Declaration:
```
function GetAppConfigDir(Global: Boolean) : string
```

Visibility: default

Description: `GetAppConfigDir` returns the name of a directory in which the application should store its configuration files on the current OS. If the parameter `Global` is `True` then the directory returned is a global directory, i.e. valid for all users on the system. If the parameter `Global` is false, then the directory is specific for the user who is executing the program. On systems that do not support multi-user environments, these two directories may be the same.

The directory which is returned is the name of the directory where the application is supposed to store files. This does not mean that the directory exists, or that the user can write in this directory (especially if `Global=True`). It just returns the name of the appropriate location. Also note that the returned name always contains an ending path delimiter.

On systems where the operating system provides a call to determine this location, this call will be used. On systems where there is no such call, an algorithm is used which reflects common practice on that system.

The application name is deduced from the binary name via the ApplicationName (1683) call, but can be configured by means of the OnGetApplicationName (1666) callback.

If VendorName (1815) is not-empty, then `VendorName` will also be inserted before the application-specific directory.

Errors: None.

See also: GetAppConfigFile (1746), ApplicationName (1683), OnGetApplicationName (1666), CreateDir (1692), SysConfigDir (1650), VendorName (1815)

76.15.144 GetAppConfigFile

Synopsis: Return an appropriate name for an application configuration file.

Declaration:
```
function GetAppConfigFile(Global: Boolean) : string
function GetAppConfigFile(Global: Boolean; SubDir: Boolean) : string
```

Visibility: default

Description: `GetAppConfigFile` returns the name of a file in which the application can store its configuration parameters. The exact name and location of the file depends on the customs of the operating system.

The `Global` parameter determines whether it is a global configuration file (value `True`) or a personal configuration file (value `False`).

The parameter `SubDir`, in case it is set to `True`, will insert the name of a directory before the filename. This can be used in case the application needs to store other data than configuration data in an application-specific directory. Default behaviour is to set this to `False`.

Note that on Windows, even when `Subdir` is `False`, a subdirectory is created for the application configuration files, as per the windows specifications. Specifying `true` will create a subdirectory of the application settings subdirectory.

The default file extension of the returned file is: `.cfg`

No assumptions should be made about the existence or writeability of this file, or the directory where the file should reside. It is best to call ForceDirectories (1735) prior to opening a file with the resulting filename.

On systems where the operating system provides a call to determine the location of configuration files, this call will be used. On systems where there is no such call, an algorithm is used which reflects common practice on that system.

The application name is deduced from the binary name via the ApplicationName (1683) call, but can be configured by means of the OnGetApplicationName (1666) callback.

If VendorName (1815) is not-empty, then `VendorName` will be inserted in the path for the config file directory.

Errors: None.

See also: GetAppConfigDir (1746), OnGetApplicationName (1666), ApplicationName (1683), CreateDir (1692), ConfigExtension (1643), SysConfigDir (1650), VendorName (1815)

76.15.145 GetCompiledArchitecture

Declaration: `function GetCompiledArchitecture : TOSVersion.TArchitecture`

Visibility: default

76.15.146 GetCompiledPlatform

Declaration: `function GetCompiledPlatform : TOSVersion.TPlatform`

Visibility: default

76.15.147 GetCurrentDir

Synopsis: Return the current working directory of the application.

Declaration: `function GetCurrentDir : AnsiString`

Visibility: default

Description: `GetCurrentDir` returns the current working directory.

Errors: None.

See also: SetCurrentDir (1770), DiskFree (1701), DiskSize (1702)

Listing: ./examples/sysutex/ex28.pp

```
Program Example28;

{ This program demonstrates the GetCurrentDir function }

Uses sysutils;

Begin
  WriteIn ('Current Directory is : ',GetCurrentDir);
End.
```

76.15.148 GetDirs

Synopsis: Return a list of directory names from a path.

Declaration:
```
function GetDirs(var DirName: UnicodeString;
                 var Dirs: Array of PUnicodeChar) : LongInt
function GetDirs(var DirName: RawByteString;
                 var Dirs: Array of PAnsiChar) : LongInt
```

Visibility: default

Description: `GetDirs` splits DirName in a null-byte separated list of directory names, `Dirs` is an array of `PChars`, pointing to these directory names. The function returns the number of directories found, or -1 if none were found. DirName must contain only OSDirSeparator as Directory separator chars.

Errors: None.

See also: ExtractRelativePath (1712)

Listing: ./examples/sysutex/ex45.pp

```
Program Example45;

{ This program demonstrates the GetDirs function }
{$H+}

Uses sysutils;

Var Dirs : Array[0..127] of pchar;
    I,Count : longint;
    Dir,NewDir : String;

Begin
  Dir:=GetCurrentDir;
  Writeln ('Dir : ',Dir);
  NewDir:='';
  count:=GetDirs(Dir,Dirs);
  For I:=0 to Count-1 do
    begin
    NewDir:=NewDir+'/'+StrPas(Dirs[I]);
    Writeln (NewDir);
    end;
End.
```

76.15.149 GetDriveIDFromLetter

Synopsis: Return the drive ID based on a drive letter.

Declaration:
```
function GetDriveIDFromLetter(const ADrive: RawByteString) : Byte
function GetDriveIDFromLetter(const ADrive: UnicodeString) : Byte
```

Visibility: default

Description: `GetDriveIDFromLetter` returns the drive ID based on a drive letter. The drive ID can then be used in the DiskSize (1702) and DiskFree (1701) calls.

See also: DiskSize (1702), DiskFree (1701)

76.15.150 GetEnvironmentString

Synopsis: Return an environment variable by index.

Declaration:
```
function GetEnvironmentString(Index: Integer) : AnsiString
```

Visibility: default

Description: `GetEnvironmentString` returns the `Index`-th environment variable. The index is 1 based, and is bounded from above by the result of GetEnvironmentVariableCount (1749).

For an example, GetEnvironmentVariableCount (1749).

Remark Note that on Windows, environment strings can start with an equal sign (=). This is a trick used to pass the current working directory to a newly created process. In this case, extracting the variable name as the characters before the first equal sign will result in an empty name.

Errors: If there is no environment, an empty string is returned.

See also: GetEnvironmentVariable (1749), GetEnvironmentVariableCount (1749)

76.15.151 GetEnvironmentVariable

Synopsis: Return the value of an environment variable.

Declaration:
```
function GetEnvironmentVariable(const EnvVar: AnsiString) : AnsiString
                                ;  Overload
function GetEnvironmentVariable(const EnvVar: UnicodeString)
                                 : UnicodeString;  Overload
```

Visibility: default

Description: `GetEnvironmentVariable` returns the value of the `EnvVar` environment variable. If the specified variable does not exist or `EnvVar` is empty, an empty string is returned.

See also: GetEnvironmentString (1749), GetEnvironmentVariableCount (1749)

76.15.152 GetEnvironmentVariableCount

Synopsis: Return the number of variables in the environment.

Declaration:
```
function GetEnvironmentVariableCount : Integer
```

Visibility: default

Description: `GetEnvironmentVariableCount` returns the number of variables in the environment. The number is 1 based, but the result may be zero if there are no environment variables.

Errors: If there is no environment, -1 may be returned.

See also: GetEnvironmentString (1749), GetEnvironmentVariable (1749)

Listing: ./examples/sysutex/ex92.pp

```
{$h+}
program example92;

{ This program demonstrates the
  GetEnvironmentVariableCount function }
```

```
uses sysutils;

Var
  I : Integer;

begin
  For I:=1 to GetEnvironmentVariableCount do
    WriteIn(i:3,' : ',GetEnvironmentString(i));
end.
```

76.15.153 GetFileAsString

Synopsis: Return the contents of a file as a string.

Declaration:
```
function GetFileAsString(const aFileName: RawByteString) : RawByteString
function GetFileAsString(const aFileName: RawByteString;
                         aEncoding: TEncoding) : RawByteString
function GetFileAsString(const aFileName: UnicodeString) : UnicodeString
function GetFileAsString(const aFileName: UnicodeString;
                         aEncoding: TEncoding) : UnicodeString
```

Visibility: default

Description: `GetFileAsString` returns the contents of `aFileName` or `aHandle` as a string. You can specify the string encoding in the `aEncoding` argument. If no encoding is specified TEncoding.SystemEncoding (1875) is used. The whole file is read in the case of `aHandle`: The file position is first set to offset zero.

Errors: If an invalid file handle or a file handle that does not support the seek operation, a EInOutError (1833) exception is raised.

See also: GetFileContents (1750), TEncoding.SystemEncoding (1875), TEncoding.GetAnsiString (1869)

76.15.154 GetFileContents

Synopsis: Return the contents of a file as an array of bytes.

Declaration:
```
function GetFileContents(const aFileName: RawByteString) : TBytes
function GetFileContents(const aFileName: UnicodeString) : TBytes
function GetFileContents(const aHandle: THandle) : TBytes
```

Visibility: default

Description: `GetFileContents` returns the contents of `aFileName` or `aHandle` as an array of bytes. The whole file is read in the case of `aHandle`: The file position is first set to offset zero.

Errors: If an invalid file handle or a file handle that does not support the seek operation, a EInOutError (1833) exception is raised.

See also: GetFileAsString (1750)

76.15.155 GetFileHandle

Synopsis: Extract OS handle from an untyped file or text file.

Declaration:
```
function GetFileHandle(var f: File) : THandle
function GetFileHandle(var f: Text) : THandle
```

Visibility: default

Description: `GetFileHandle` returns the operating system handle for the file descriptor `F`. It can be used in various file operations which are not directly supported by the pascal language.

76.15.156 GetLastOSError

Synopsis: Return the last code from the OS.

Declaration:
```
function GetLastOSError : Integer
```

Visibility: default

Description: `GetLastOSError` returns the error code from the last operating system call. It does not reset this code. In general, it should be called when an operating system call reported an error condition. In that case, `GetLastOSError` gives extended information about the error.

No assumptions should be made about the resetting of the error code by subsequent OS calls. This may be platform dependent.

See also: RaiseLastOSError (1766)

76.15.157 GetLocalTime

Synopsis: Get the local time.

Declaration:
```
procedure GetLocalTime(var SystemTime: TSystemTime)
```

Visibility: default

Description: `GetLocalTime` returns the system time in a TSystemTime (1662) format.

Errors: None.

See also: Now (1765), Date (1694), Time (1801), TSystemTime (1662)

76.15.158 GetLocalTimeOffset

Synopsis: Return local timezone offset.

Declaration:
```
function GetLocalTimeOffset : Integer
function GetLocalTimeOffset(const DateTime: TDateTime;
                            const InputIsUTC: Boolean;
                            out Offset: Integer) : Boolean
function GetLocalTimeOffset(const DateTime: TDateTime;
                            const InputIsUTC: Boolean) : Integer
```

Visibility: default

Description: `GetLocalTimeOffset` returns the local timezone offset in minutes. This is the difference between UTC time and local time:

```
UTC = LocalTime + GetLocalTimeOffset
```

The TDateTime-overloads return the offset at the specified DateTime. In case the TDateTime-aware function is not supported on the current platform, False is returned or it falls back to the offset to current time (depending on the overload used).

Note that on Linux/Unix, the information returned from the TDateTime-unaware function may be inaccurate around the DST time changes (for optimization). In that case, the unix.ReReadLocalTime (2166) unit must be used to re-initialize the timezone information.

The TDateTime-aware overloads are supported currently only on Windows Vista and newer.

See also: unix.ReReadLocalTime (2166), Date (1694), Time (1801), Now (1765)

76.15.159 GetModuleName

Synopsis: Return the name of the current module.

Declaration:
```
function GetModuleName(Module: HMODULE) : string
```

Visibility: default

Description: `GetModuleName` returns the name of the current module. On windows, this is the name of the executable when executed in an executable, or the name of the library when executed in a library.

On all other platforms, the result is always empty, since they provide no such functionality.

76.15.160 GetTempDir

Synopsis: Return name of system's temporary directory.

Declaration:
```
function GetTempDir(Global: Boolean) : string
function GetTempDir : string
```

Visibility: default

Description: `GetTempDir` returns the temporary directory of the system. If `Global` is `True` (the default value) it returns the system temporary directory, if it is `False` then a directory private to the user is returned. The returned name will end with a directory delimiter character.

These directories may be the same. No guarantee is made that this directory exists or is writeable by the user.

The OnGetTempDir (1666) handler may be set to provide custom handling of this routine: One could implement callbacks which take into consideration frameworks like KDE or GNOME, and return a different value from the default system implementation.

Errors: On error, an empty string is returned.

See also: OnGetTempDir (1666), GetTempFileName (1752)

76.15.161 GetTempFileName

Synopsis: Return the name of a temporary file.

Declaration:
```
function GetTempFileName(const Dir: string; const Prefix: string)
                         : string
function GetTempFileName : string
function GetTempFileName(Dir: PChar; Prefix: PChar; uUnique: DWord;
                         TempFileName: PChar) : DWord
```

Visibility: default

Description: `GetTempFileName` returns the name of a temporary file in directory `Dir`. The name of the file starts with `Prefix`.

If `Dir` is empty, the value returned by `GetTempDir` is used, and if `Prefix` is empty, 'TMP' is used.

The OnGetTempFile (1666) handler may be set to provide custom handling of this routine: One could implement callbacks which take into consideration frameworks like KDE or GNOME, and return a different value from the default system implementation.

Errors: On error, an empty string is returned.

See also: GetTempDir (1752), OnGetTempFile (1666)

76.15.162 GetTickCount

Synopsis: Get tick count (32-bit, deprecated).

Declaration: `function GetTickCount : LongWord`

Visibility: default

Description: `GetTickCount` returns an increasing clock tick count in milliseconds. It is useful for time measurements, but no assumptions should be made as to the interval between the ticks. This function is provided for Delphi compatibility, use GetTickCount64 (1753) instead.

See also: GetTickCount64 (1753), Now (1765), Time (1801), Sleep (1771)

76.15.163 GetTickCount64

Synopsis: Get tick count (64-bit).

Declaration: `function GetTickCount64 : QWord`

Visibility: default

Description: `GetTickCount64` returns an increasing clock tick count in milliseconds. It is useful for time measurements, but no assumptions should be made as to the interval between the ticks.

See also: Now (1765), Time (1801), Sleep (1771)

76.15.164 GetUniversalTime

Declaration: `function GetUniversalTime(var SystemTime: TSystemTime) : Boolean`

Visibility: default

76.15.165 GetUserDir

Synopsis: Returns the current user's home directory.

Declaration: `function GetUserDir : string`

Visibility: default

Description: `GetUserDir` returns the home directory of the current user. On Unix-like systems (that includes Mac OS X), this is the value of the HOME environment variable. On Windows, this is the PROFILE special folder. On all other platforms, the application installation directory is returned.

If non-empty, it contains a trailing path delimiter.

See also: GetAppConfigDir (1746)

76.15.166 GuidCase

Synopsis: Return the index of a GUID in an array of GUID values.

Declaration:
```
function GuidCase(const GUID: TGuid; const List: Array of TGuid)
                        : Integer
```

Visibility: default

Description: `GuidCase` returns the index of `GUID` in the array `List`, where 0 denotes the first element in the list. If GUID is not present in the list, -1 is returned.

See also: IsEqualGUID (1759)

76.15.167 GUIDToString

Synopsis: Convert a `TGUID` to a string representation.

Declaration:
```
function GUIDToString(const GUID: TGuid) : string
```

Visibility: default

Description: `GUIDToString` converts the GUID identifier in `GUID` to a string representation in the form

```
{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}
```

Where each `X` is a hexadecimal digit.

Errors: None.

See also: Supports (1798), #rtl.system.TGUID (1420), StringToGUID (1779), IsEqualGuid (1759)

76.15.168 HashName

Synopsis: Calculate a hash from a null-terminated string.

Declaration:
```
function HashName(Name: PAnsiChar) : LongWord
```

Visibility: default

Description: `HashName` calculates a hash value from a null terminated string. The hash value is calculated in such a way that it returns the same value for strings that only differ in case.

76.15.169 HookSignal

Synopsis: Hook a specified signal.

Declaration:
```
procedure HookSignal(RtlSigNum: Integer)
```

Visibility: default

Description: `HookSignal` installs the RTL default signal handler for signal `RtlSigNum`. It does not check whether the signal is already handled, and should therefor only be called if `InquireSignal` returns `ssNotHooked`.

76.15.170 IncAMonth

Synopsis: Increase a date with a certain amount of months.

Declaration:
```
procedure IncAMonth(var Year: Word; var Month: Word; var Day: Word;
                    NumberOfMonths: Integer)
```

Visibility: default

Description: `IncAMonth` increases the date as specified by `Year`, `Month`, `Day` with `NumberOfMonths`. It takes care of the number of days in a month when calculating the result.

This function does the same as IncMonth (1756), but operates on an already decoded date.

See also: IncMonth (1756)

76.15.171 IncludeLeadingPathDelimiter

Synopsis: Prepend a path delimiter if there is not already one.

Declaration:
```
function IncludeLeadingPathDelimiter(const Path: UnicodeString)
                                     : UnicodeString
function IncludeLeadingPathDelimiter(const Path: RawByteString)
                                     : RawByteString
```

Visibility: default

Description: `IncludeLeadingPathDelimiter` will insert a path delimiter (#rtl.system.DirectorySeparator (1371)) in the first position of `Path`, if there is not already a directory separator at that position. It will return the resulting string. If the path is empty, a `DirectorySeparator` character is returned.

See also: IncludeTrailingPathDelimiter (1756), ExcludeLeadingPathDelimiter (1706), ExcludeTrailingPathDelimiter (1707), ConcatPaths (1691)

Listing: ./examples/sysutex/ex94.pp

```
Program Example94;

{ This program demonstrates the IncludeLeadingPathDelimiter function }

Uses sysutils;

Begin
End.
```

76.15.172 IncludeTrailingBackslash

Synopsis: Add trailing directory separator to a pathname, if needed.

Declaration:
```
function IncludeTrailingBackslash(const Path: UnicodeString)
                                  : UnicodeString
function IncludeTrailingBackslash(const Path: RawByteString)
                                  : RawByteString
```

Visibility: default

Description: `IncludeTrailingBackslash` is provided for backwards compatibility with Delphi. Use IncludeTrailingPathDelimiter (1756) instead.

See also: IncludeTrailingPathDelimiter (1756), ExcludeTrailingPathDelimiter (1707), PathDelim (1648), IsPathDelimiter (1761)

76.15.173 IncludeTrailingPathDelimiter

Synopsis: Add trailing directory separator to a pathname, if needed.

Declaration:
```
function IncludeTrailingPathDelimiter(const Path: UnicodeString)
                                      : UnicodeString
function IncludeTrailingPathDelimiter(const Path: RawByteString)
                                      : RawByteString
```

Visibility: default

Description: `IncludeTrailingPathDelimiter` adds a trailing path delimiter character (PathDelim (1648)) to `Path` if none is present yet, and returns the result.

If `Path` is empty, a path delimiter is returned, for Delphi compatibility.

See also: IncludeTrailingBackslash (1756), ExcludeTrailingPathDelimiter (1707), PathDelim (1648), IsPathDelimiter (1761)

76.15.174 IncMonth

Synopsis: Increases the month in a `TDateTime` value with a given amount.

Declaration:
```
function IncMonth(const DateTime: TDateTime; NumberOfMonths: Integer)
                  : TDateTime
```

Visibility: default

Description: `IncMonth` increases the month number in `DateTime` with `NumberOfMonths`. It wraps the result as to get a month between 1 and 12, and updates the year accordingly. `NumberOfMonths` can be negative, and can be larger than 12 (in absolute value).

Errors: None.

See also: Date (1694), Time (1801), Now (1765)

Listing: ./examples/sysutex/ex15.pp

```
Program Example15;

{ This program demonstrates the IncMonth function }

Uses sysutils;

Var ThisDay : TDateTime;

Begin
  ThisDay:=Date;
  Writeln ('ThisDay : ',DateToStr(ThisDay));
  Writeln ('6 months ago :',DateToStr(IncMonth(ThisDay,-6)));
  Writeln ('6 months from now :' ,DateToStr(IncMonth(ThisDay,6)));
  Writeln ('12 months ago :',DateToStr(IncMonth(ThisDay,-12)));
  Writeln ('12 months from now :' ,DateToStr(IncMonth(ThisDay,12)));
  Writeln ('18 months ago :',DateToStr(IncMonth(ThisDay,-18)));
  Writeln ('18 months from now :' ,DateToStr(IncMonth(ThisDay,18)));
End.
```

76.15.175 InquireSignal

Synopsis: Check whether a signal handler is set (unix only).

Declaration:
```
function InquireSignal(RtlSigNum: Integer) : TSignalState
```

Visibility: default

Description: `RtlSigNum` will check whether the signal `RtlSigNum` is being handled, and by whom. It returns a `TSignalState` result to report the state of the signal, which can be one of the following values:

ssNotHookedNo signal handler is set for the signal.

ssHookedA signal handler is set by the RTL code for the signal.

ssOverriddenA signal handler was set for the signal by third-party code.

This routine works by resetting the signal handlers, so it is risky to call.

76.15.176 IntToHex

Synopsis: Convert an integer value to a hexadecimal string.

Declaration:
```
function IntToHex(Value: LongInt; Digits: Integer) : string
function IntToHex(Value: Int64; Digits: Integer) : string
function IntToHex(Value: QWord; Digits: Integer) : string
function IntToHex(Value: Int8) : string
function IntToHex(Value: UInt8) : string
function IntToHex(Value: Int16) : string
function IntToHex(Value: UInt16) : string
function IntToHex(Value: Int32) : string
function IntToHex(Value: UInt32) : string
function IntToHex(Value: Int64) : string
function IntToHex(Value: UInt64) : string
```

Visibility: default

Description: `IntToHex` converts `Value` to a hexadecimal string representation. The result will contain at least `Digits` characters. If `Digits` is less than the needed number of characters, the string will NOT be truncated. If `Digits` is larger than the needed number of characters, the result is padded with zeroes.

Errors: None.

See also: IntToStr (1758)

Listing: ./examples/sysutex/ex73.pp

```
Program Example73;

{ This program demonstrates the IntToHex function }

Uses sysutils;

Var l : longint;

Begin
  For l:=0 to 31 do
      begin
      WriteLn (IntToHex(1 shl l,8));
      WriteLn (IntToHex(15 shl l,8))
      end;
End.
```

76.15.177 IntToStr

Synopsis: Convert an integer value to a decimal string.

Declaration:
```
function IntToStr(Value: LongInt) : string
function IntToStr(Value: Int64) : string
function IntToStr(Value: QWord) : string
```

Visibility: default

Description: `IntToStr` converts `Value` to its string representation. The resulting string has only as much characters as needed to represent the value. If the value is negative a minus sign is prepended to the string.

Errors: None.

See also: IntToHex (1757), StrToInt (1793), UIntToStr (1810)

Listing: ./examples/sysutex/ex74.pp

```
Program Example74;

{ This program demonstrates the IntToStr function }

Uses sysutils;

Var l : longint;

Begin
  For l:=0 to 31 do
      begin
```

```
      WriteLn (IntToStr(1 shl I));
      WriteLn (IntToStr(15 shl I));
      end;
End.
```

76.15.178 IsDelimiter

Synopsis: Check whether a given string is a delimiter character.

Declaration:
```
function IsDelimiter(const Delimiters: string; const S: string;
                     Index: SizeInt) : Boolean
```

Visibility: default

Description: `IsDelimiter` checks whether the `Index`-th character in the string `S` is a delimiter character as passed in `Delimiters`. If `Index` is out of range, `False` is returned.

Errors: None.

See also: LastDelimiter (1762)

76.15.179 IsEqualGUID

Synopsis: Check whether two `TGUID` variables are equal.

Declaration:
```
function IsEqualGUID(const guid1: TGuid; const guid2: TGuid) : Boolean
```

Visibility: default

Description: `IsEqualGUID` checks whether `guid1` and `guid2` are equal, and returns `True` if this is the case, or `False` otherwise.

See also: Supports (1798), #rtl.system.TGUID (1420), StringToGUID (1779), GuidToString (1754)

76.15.180 IsFileNameCasePreserving

Declaration:
```
function IsFileNameCasePreserving(const aFileName: RawByteString)
                                  : Boolean
function IsFileNameCasePreserving(const aFileName: UnicodeString)
                                  : Boolean
```

Visibility: default

76.15.181 IsFileNameCaseSensitive

Declaration:
```
function IsFileNameCaseSensitive(const aFileName: RawByteString)
                                 : Boolean
function IsFileNameCaseSensitive(const aFileName: UnicodeString)
                                 : Boolean
```

Visibility: default

76.15.182 IsLeadChar

Synopsis: Is the character code the lead character of a multi-byte character.

Declaration:
```
function IsLeadChar(C: AnsiChar) : Boolean;  Overload
function IsLeadChar(B: Byte) : Boolean;  Overload
function IsLeadChar(Ch: WideChar) : Boolean;  Overload
```

Visibility: default

Description: `IsLeadChar` returns `True` if the character `C` (or byte `B`) is the lead character (or byte) of a multi-byte character. In case of `Ch` a UTF-32 lead character.

Errors: None.

76.15.183 IsLeapYear

Synopsis: Determine whether a year is a leap year.

Declaration:
```
function IsLeapYear(Year: Word) : Boolean
```

Visibility: default

Description: `IsLeapYear` returns `True` if `Year` is a leap year, `False` otherwise.

Errors: None.

See also: IncMonth (1756), Date (1694)

Listing: ./examples/sysutex/ex16.pp

```
Program Example16;

{ This program demonstrates the IsLeapYear function }

Uses sysutils;

Var YY,MM,dd : Word;

Procedure TestYear (Y : Word);

begin
  WriteIn (Y,' is leap year : ',IsLeapYear(Y));
end;

Begin
  DeCodeDate(Date,YY,mm,dd);
  TestYear(yy);
  TestYear(2000);
  TestYear(1900);
  TestYear(1600);
  TestYear(1992);
  TestYear(1995);
End.
```

76.15.184 IsPathDelimiter

Synopsis: Is the character at the given position a pathdelimiter ?

Declaration:
```
function IsPathDelimiter(const Path: UnicodeString; Index: Integer)
                         : Boolean
function IsPathDelimiter(const Path: RawByteString; Index: Integer)
                         : Boolean
```

Visibility: default

Description: `IsPathDelimiter` returns `True` if the character at position `Index` equals PathDelim (1648), i.e. if it is a path delimiter character for the current platform.

Errors: IncludeTrailingPathDelimiter (1756)ExcludeTrailingPathDelimiter (1707)PathDelim (1648)

76.15.185 IsValidIdent

Synopsis: Check whether a string is a valid identifier name.

Declaration:
```
function IsValidIdent(const Ident: string; AllowDots: Boolean;
                      StrictDots: Boolean) : Boolean
```

Visibility: default

Description: `IsValidIdent` returns `True` if `Ident` can be used as a component name. It returns `False` otherwise. `Ident` must consist of a letter or underscore, followed by a combination of letters, numbers or underscores to be a valid identifier.

Errors: None.

Listing: ./examples/sysutex/ex75.pp

```
Program Example75;

{ This program demonstrates the IsValidIdent function }

Uses sysutils;

Procedure Testit (S : String);

begin
  Write ('"',S,'" is ');
  If not IsVAlidIdent(S) then
    Write('NOT ');
  WriteIn ('a valid identifier');
end;

Begin
  Testit ('_MyObj');
  Testit ('My__Obj1');
  Testit ('My_1_Obj');
  Testit ('1MyObject');
  Testit ('My@Object');
  Testit ('M123');
End.
```

76.15.186 LastDelimiter

Synopsis: Return the last occurrence of a set of delimiters in a string.

Declaration:
```
function LastDelimiter(const Delimiters: string; const S: string)
                       : SizeInt
```

Visibility: default

Description: `LastDelimiter` returns the *last* occurrence of any character in the set `Delimiters` in the string `S`.

Listing: ./examples/sysutex/ex88.pp

```
Program example88;

{ This program demonstrates the LastDelimiter function }

uses SysUtils;

begin
  WriteLn(LastDelimiter('\.:','c:\filename.ext'));
end.
```

76.15.187 LeftStr

Synopsis: Return a number of characters starting at the left of a string.

Declaration:
```
function LeftStr(const S: string; Count: Integer) : string
```

Visibility: default

Description: `LeftStr` returns the `Count` leftmost characters of `S`. It is equivalent to a call to `Copy(S,1,Count)`.

Errors: None.

See also: RightStr (1768), TrimLeft (1803), TrimRight (1804), Trim (1803)

Listing: ./examples/sysutex/ex76.pp

```
Program Example76;

{ This program demonstrates the LeftStr function }

Uses sysutils;

Begin
  WriteLn (LeftStr('abcdefghijklmnopqrstuvwxyz',20));
  WriteLn (LeftStr('abcdefghijklmnopqrstuvwxyz',15));
  WriteLn (LeftStr('abcdefghijklmnopqrstuvwxyz',1));
  WriteLn (LeftStr('abcdefghijklmnopqrstuvwxyz',200));
End.
```

76.15.188 ListIndexError

Declaration:
```
procedure ListIndexError(aIndex: Integer; aMax: Integer; aObj: TObject)
```

Visibility: default

76.15.189 LoadStr

Synopsis: Load a string from the resource tables.

Declaration:
```
function LoadStr(Ident: Integer) : string
```

Visibility: default

Description: This function is not yet implemented. resources are not yet supported.

76.15.190 LocalTimeToUniversal

Declaration:
```
function LocalTimeToUniversal(LT: TDateTime) : TDateTime
function LocalTimeToUniversal(LT: TDateTime; TZOffset: Integer)
                                : TDateTime
```

Visibility: default

76.15.191 LowerCase

Synopsis: Return a lowercase version of a string.

Declaration:
```
function LowerCase(const s: string) : string;  Overload
function LowerCase(const s: string; LocaleOptions: TLocaleOptions)
                   : string;  Overload
function LowerCase(const V: variant) : string;  Overload
function LowerCase(const s: UnicodeString) : UnicodeString;  Overload
```

Visibility: default

Description: `LowerCase` returns the lowercase equivalent of `S`. Ansi characters are not taken into account, only ASCII codes below 127 are converted. It is completely equivalent to the lowercase function of the system unit, and is provided for compatibility only.

`Lowercase` does not change the number of characters (or bytes) in an ansistring or shortstring.

Errors: None.

See also: AnsiLowerCase (1674), UpperCase (1814), AnsiUpperCase (1682)

Listing: ./examples/sysutex/ex77.pp

```
Program Example77;

{ This program demonstrates the LowerCase function }

Uses sysutils;

Begin
  Writeln (LowerCase('THIS WILL COME out all LoWeRcAsE !'));
End.
```

76.15.192 MSecsToTimeStamp

Synopsis: Convert a number of milliseconds to a `TDateTime` value.

Declaration:
```
function MSecsToTimeStamp(MSecs: Comp) : TTimeStamp
function MSecsToTimeStamp(MSecs: Int64) : TTimeStamp
```

Visibility: default

Description: `MSecsToTimeStamp` converts the given number of milliseconds to a `TTimeStamp` date/time notation.

Use `TTimeStamp` variables if you need to keep very precise track of time.

Errors: None.

See also: TimeStampToMSecs (1802), DateTimeToTimeStamp (1697)

Listing: ./examples/sysutex/ex17.pp

```
Program Example17;

{ This program demonstrates the MSecsToTimeStamp function }

Uses sysutils;

Var MS : Comp;
    TS : TTimeStamp;
    DT : TDateTime;
Begin
  TS:=DateTimeToTimeStamp(Now);
  Writeln ('Now in days since 1/1/0001      : ',TS.Date);
  Writeln ('Now in millisecs since midnight : ',TS.Time);
  MS:=TimeStampToMSecs(TS);
  Writeln ('Now in millisecs since 1/1/0001 : ',MS);
  MS:=MS-1000*3600*2;
  TS:=MSecsToTimeStamp(MS);
  DT:=TimeStampToDateTime(TS);
  Writeln ('Now minus 1 day : ',DateTimeToStr(DT));
End.
```

76.15.193 NewStr

Synopsis: Allocate a new ansistring on the heap.

Declaration: `function NewStr(const S: string) : PString;  Overload`

Visibility: default

Description: `NewStr` assigns a new dynamic string on the heap, copies `S` into it, and returns a pointer to the newly assigned string.

This function is obsolete, and shouldn't be used any more. The `AnsiString` mechanism also allocates ansistrings on the heap, and should be preferred over this mechanism.

For an example, see AssignStr (1684).

Errors: If not enough memory is present, an EOutOfMemory exception will be raised.

See also: AssignStr (1684), DisposeStr (1703)

76.15.194 Now

Synopsis: Returns the current date and time.

Declaration: `function Now : TDateTime`

Visibility: default

Description: `Now` returns the current date and time. It is equivalent to `Date+Time`.

Errors: None.

See also: Date (1694), Time (1801)

Listing: ./examples/sysutex/ex18.pp

```
Program Example18;

{ This program demonstrates the Now function }

Uses sysutils;

Begin
  WriteLn ('Now : ',DateTimeToStr(Now));
End.
```

76.15.195 NowUTC

Declaration: `function NowUTC : TDateTime`

Visibility: default

76.15.196 OutOfMemoryError

Synopsis: Raise an `EOutOfMemory` exception.

Declaration: `procedure OutOfMemoryError`

Visibility: default

Description: `OutOfMemoryError` raises an EOutOfMemory (1836) exception, with an exception object that has been allocated on the heap at program startup. The program should never create an EOutOfMemory (1836) exception, but always call this routine.

See also: EOutOfMemory (1836)

76.15.197 QuotedStr

Synopsis: Return a quotes version of a string.

Declaration: `function QuotedStr(const S: string) : string`

Visibility: default

Description: `QuotedStr` returns the string `S`, quoted with single quotes. This means that `S` is enclosed in single quotes, and every single quote in `S` is doubled. It is equivalent to a call to `AnsiQuotedStr(s, "")`.

Errors: None.

See also: AnsiQuotedStr (1675), AnsiExtractQuotedStr (1672)

Listing: ./examples/sysutex/ex78.pp

```
Program Example78;

{ This program demonstrates the QuotedStr function }

Uses sysutils;

Var S : AnsiString;

Begin
  S:='He said ''Hello'' and walked on';
  Writeln (S);
  Writeln ('  becomes');
  Writeln (QuotedStr(S));
End.
```

76.15.198 RaiseLastOSError

Synopsis: Raise an exception with the last Operating System error code.

Declaration:
```
procedure RaiseLastOSError;  Overload
procedure RaiseLastOSError(LastError: Integer);  Overload
```

Visibility: default

Description: `RaiseLastOSError` raises an EOSError (1836) exception with the error code returned by `GetLastOSError`. If the Error code is nonzero, then the corresponding error message will be returned. If the error code is zero, a standard message will be returned.

Errors: This procedure may not be implemented on all platforms. If it is not, then a normal Exception (1839) will be raised.

See also: EOSError (1836), GetLastOSError (1751), Exception (1839)

76.15.199 RemoveDir

Synopsis: Remove a directory from the file system.

Declaration:
```
function RemoveDir(const Dir: RawByteString) : Boolean
function RemoveDir(const Dir: UnicodeString) : Boolean
```

Visibility: default

Description: `RemoveDir` removes directory `Dir` from the disk. If the directory is not absolute, it is appended to the current working directory.

For an example, see CreateDir (1692).

Errors: In case of error (e.g. the directory isn't empty) the function returns `False`. If successful, `True` is returned.

76.15.200 RenameFile

Synopsis: Rename a file.

Declaration:
```
function RenameFile(const OldName: UnicodeString;
                    const NewName: UnicodeString) : Boolean
function RenameFile(const OldName: RawByteString;
                    const NewName: RawByteString) : Boolean
```

Visibility: default

Description: `RenameFile` renames a file from `OldName` to `NewName`. The function returns `True` if successful, `False` otherwise. For safety, the new name must be a full path specification, including the directory, otherwise it will be assumed to be a filename relative to the current working directory. *Remark:* The implementation of `RenameFile` relies on the underlying OS's support for renaming/moving a file. Whether or not a file can be renamed across disks or partitions depends entirely on the OS. On unix-like OS-es, the rename function will fail when used across partitions. On Windows, it will work.

Errors: On Error, `False` is returned.

See also: DeleteFile (1700)

Listing: ./examples/sysutex/ex44.pp

```
Program Example44;

{ This program demonstrates the RenameFile function }

Uses sysutils;

Var F : Longint;
    S : String;

Begin
  S:='Some short file.';
  F:=FileCreate ('test.dap');
  FileWrite(F,S[1],Length(S));
  FileClose(F);
  If RenameFile ('test.dap','test.dat') then
    WriteIn ('Successfully renamed files.');
End.
```

76.15.201 ReplaceDate

Synopsis: Replace the date part of a date/time stamp.

Declaration:
```
procedure ReplaceDate(var DateTime: TDateTime; const NewDate: TDateTime)
```

Visibility: default

Description: `ReplaceDate` replaces the date part of `DateTime` with `NewDate`. The time part is left unchanged.

See also: ReplaceTime (1768)

76.15.202 ReplaceTime

Synopsis: Replace the time part.

Declaration: `procedure ReplaceTime(var dati: TDateTime; NewTime: TDateTime)`

Visibility: default

Description: `ReplaceTime` replaces the time part in `dati` with `NewTime`. The date part remains untouched.

76.15.203 RightStr

Synopsis: Return a number of characters from a string, starting at the end.

Declaration: `function RightStr(const S: string; Count: Integer) : string`

Visibility: default

Description: `RightStr` returns the `Count` rightmost characters of `S`. It is equivalent to a call to `Copy(S,Length(S)+1-Count,Coun`
If `Count` is larger than the actual length of `S` only the real length will be used.

Errors: None.

See also: LeftStr (1762), Trim (1803), TrimLeft (1803), TrimRight (1804)

Listing: ./examples/sysutex/ex79.pp

```
Program Example79;

{ This program demonstrates the RightStr function }

Uses sysutils;

Begin
  WriteLn (RightStr('abcdefghijklmnopqrstuvwxyz',20));
  WriteLn (RightStr('abcdefghijklmnopqrstuvwxyz',15));
  WriteLn (RightStr('abcdefghijklmnopqrstuvwxyz',1));
  WriteLn (RightStr('abcdefghijklmnopqrstuvwxyz',200));
End.
```

76.15.204 SafeFormat

Declaration:
```
function SafeFormat(const Fmt: AnsiString; Args: Array of const;
                    const FormatSettings: TFormatSettings) : UTF8String
                    ;  Overload
function SafeFormat(const Fmt: AnsiString; Args: Array of const)
                     : UTF8String;  Overload
```

Visibility: default

76.15.205 SafeLoadLibrary

Synopsis: Load a library safely.

Declaration:
```
function SafeLoadLibrary(const FileName: AnsiString; ErrorMode: DWord)
                         : HMODULE
```

Visibility: default

Description: `SafeLoadLibrary` saves and restores some registers before and after issuing a call to `LoadLibrary`.

Errors: None.

76.15.206 SameFileName

Synopsis: Are two filenames referring to the same file ?

Declaration:
```
function SameFileName(const S1: string; const S2: string) : Boolean
```

Visibility: default

Description: `SameFileName` returns `True` if calling AnsiCompareFileName (1670) with arguments `S1` and `S2` returns 0, and returns `False` otherwise.

Errors: None.

See also: AnsiCompareFileName (1670)

76.15.207 SameStr

Synopsis: Check whether 2 strings are the same, case insensitive.

Declaration:
```
function SameStr(const s1: string; const s2: string) : Boolean
                 ;  Overload
function SameStr(const s1: string; const s2: string;
                 LocaleOptions: TLocaleOptions) : Boolean;  Overload
```

Visibility: default

Description: `SameStr` checks whether `S1` and `S2` are the same. This is equivalent to checking that CompareStr (1689) returns 0.

Errors: None.

See also: CompareStr (1689)

76.15.208 SameText

Synopsis: Checks whether 2 strings are the same (case insensitive).

Declaration:
```
function SameText(const s1: string; const s2: string) : Boolean
                  ;  Overload
function SameText(const s1: string; const s2: string;
                  LocaleOptions: TLocaleOptions) : Boolean;  Overload
```

Visibility: default

Description: `SameText` calls CompareText (1690) with `S1` and `S2` as parameters and returns `True` if the result of that call is zero, or `False` otherwise.

Errors: None.

See also: CompareText (1690), AnsiSameText (1675), AnsiSameStr (1675)

76.15.209 SetCurrentDir

Synopsis: Set the current directory of the application.

Declaration:
```
function SetCurrentDir(const NewDir: RawByteString) : Boolean
function SetCurrentDir(const NewDir: UnicodeString) : Boolean
```

Visibility: default

Description: `SetCurrentDir` sets the current working directory of your program to `NewDir`. It returns `True` if the function was successful, `False` otherwise.

Errors: In case of error, `False` is returned.

See also: GetCurrentDir (1747)

76.15.210 SetDirSeparators

Synopsis: Set the directory separators to the known directory separators.

Declaration:
```
function SetDirSeparators(const FileName: UnicodeString) : UnicodeString
function SetDirSeparators(const FileName: RawByteString) : RawByteString
```

Visibility: default

Description: `SetDirSeparators` returns `FileName` with all possible DirSeparators replaced by `OSDirSeparator`.

Errors: None.

See also: ExpandFileName (1708), ExtractFilePath (1712), ExtractFileDir (1710)

Listing: ./examples/sysutex/ex47.pp

```
Program Example47;

{ This program demonstrates the SetDirSeparators function }

Uses sysutils;

Begin
  Writeln (SetDirSeparators('/pp\bin/win32\ppc386'));
End.
```

76.15.211 ShowException

Synopsis: Show the current exception to the user.

Declaration:
```
procedure ShowException(ExceptObject: TObject; ExceptAddr: Pointer)
```

Visibility: default

Description: `ShowException` shows a message stating that a `ExceptObject` was raised at address `ExceptAddr`. It uses ExceptionErrorMessage (1706) to create the message, and is aware of the fact whether the application is a console application or a GUI application. For a console application, the message is written to standard error output. For a GUI application, OnShowException (1666) is executed.

Errors: If, for a GUI application, OnShowException (1666) is not set, no message will be displayed to the user.

The exception message can be at most 255 characters long: It is possible that no memory can be allocated on the heap, so ansistrings are not available, so a shortstring is used to display the message.

See also: ExceptObject (1706), ExceptAddr (1705), ExceptionErrorMessage (1706)

76.15.212 Sleep

Synopsis: Suspend execution of a program for a certain time.

Declaration:
```
procedure Sleep(milliseconds: Cardinal)
```

Visibility: default

Description: `Sleep` suspends the execution of the program for the specified number of milliseconds (`milliseconds`). After the specified period has expired, program execution resumes.

Remark The indicated time is not exact, i.e. it is a minimum time. No guarantees are made as to the exact duration of the suspension.

76.15.213 SScanf

Synopsis: Scan a string for substrings and return the content of these substrings as typed values.

Declaration:
```
function SScanf(const s: string; const fmt: string;
                const Pointers: Array of Pointer) : Integer
```

Visibility: default

Description: `SScanF` does in essence the opposite of Format (1735): it scans the string `S` for the elements specified in `Fmt`, and returns the value of the found elements in the memory locations pointed to by the addresses in `Pointers`. The `Fmt` can contain placeholders of the form `%X` where `X` can be one of the following characters:

dPlaceholder for a decimal number.

fPlaceholder for a floating point number (an extended)

sPlaceholder for a string of arbitrary length.

cPlaceholder for a single character

The `Pointers` array contains a list of pointers, each pointer should point to a memory location of a type that corresponds to the type of placeholder in that position:

dA pointer to an integer.

fA pointer to an extended.

sA pointer to an ansistring.

cA pointer to a single character.

On return, these locations will be filled with the actual values found in `S` for the placeholders in `fmt`. The return value of the function is the number of found values.

Errors: No error checking is performed on the type of the memory location.

See also: Format (1735)

Listing: ./examples/sysutex/ex98.pp

```
Program Example98;
{$mode objfpc}
{$h+}
{ This program demonstrates the  function }

Uses sysutils;

var
  count,i: Integer;
  f: Extended;
  s: String;

begin
  count:=SScanf('234 32.4 hello','%d %f %s',[@i,@f,@s]);
  writeln(count,' ',i,' ',f,' ',s);
End.
```

76.15.214 StrAlloc

Synopsis: Allocate a null-terminated string on the heap.

Declaration:
```
function StrAlloc(Size: Cardinal) : PChar
```

Visibility: default

Description: `StrAlloc` reserves memory on the heap for a string with length `Len`, terminating `#0` included, and returns a pointer to it.

Additionally, `StrAlloc` allocates 4 extra bytes to store the size of the allocated memory. Therefore this function is NOT compatible with the StrAlloc (1310) function of the `Strings` unit.

For an example, see StrBufSize (1772).

Errors: None.

See also: StrBufSize (1772), StrDispose (1775), StrAlloc (1310)

76.15.215 StrBufSize

Synopsis: Return the size of a null-terminated string allocated on the heap.

Declaration:
```
function StrBufSize(Str: PChar) : Cardinal
function StrBufSize(str: PWideChar) : Cardinal
```

Visibility: default

Description: `StrBufSize` returns the memory allocated for `Str`. This function ONLY gives the correct result if `Str` was allocated using StrAlloc (1772).

Errors: If no more memory is available, a runtime error occurs.

See also: StrAlloc (1772), StrDispose (1775)

Listing: ./examples/sysutex/ex46.pp

```
Program Example46;

{ This program demonstrates the StrBufSize function }
{$H+}

Uses sysutils;

Const S = 'Some nice string';

Var P : Pchar;

Begin
  P:=StrAlloc(Length(S)+1);
  StrPCopy(P,S);
  Write (P, ' has length ',length(S));
  WriteIn (' and  buffer size ',StrBufSize(P));
  StrDispose(P);
End.
```

76.15.216 StrByteType

Synopsis: Return the type of byte in a null-terminated string for a multi-byte character set.

Declaration:
```
function StrByteType(Str: PChar; Index: SizeUInt) : TMbcsByteType
```

Visibility: default

Description: `StrByteType` returns the type of byte in the null-terminated string `Str` at (0-based) position `Index`.

Errors: No checking on the index is performed.

See also: TMbcsByteType (1659), ByteType (1687)

76.15.217 strcat

Synopsis: Concatenate 2 null-terminated strings.

Declaration:
```
function strcat(dest: PChar; source: PChar) : PChar
function strcat(dest: PWideChar; source: PWideChar) : PWideChar
```

Visibility: default

Description: Attaches `Source` to `Dest` and returns `Dest`.

Errors: No length checking is performed.

See also: StrLCat (1779)

Listing: ./examples/stringex/ex11.pp

```
Program Example11;

Uses strings;

{ Program to demonstrate the StrCat function. }
```

```
Const P1 : PChar = 'This is a PChar String.';

Var P2 : PChar;

begin
  P2:=StrAlloc (StrLen(P1)*2+1);
  StrMove (P2,P1,StrLen(P1)+1); { P2=P1 }
  StrCat (P2,P1);               { Append P2 once more }
  WriteIn ('P2 : ',P2);
  StrDispose(P2);
end.
```

76.15.218 StrCharLength

Synopsis: Return the length of a null-terminated string in characters.

Declaration:
```
function StrCharLength(const Str: PChar) : SizeInt
```

Visibility: default

Description: `StrCharLength` returns the length of the null-terminated string `Str` (a widestring) in characters (not in bytes). It uses the widestring manager to do this.

76.15.219 strcomp

Synopsis: Compare 2 null-terminated strings, case sensitive.

Declaration:
```
function strcomp(str1: PChar; str2: PChar) : SizeInt
function strcomp(str1: PWideChar; str2: PWideChar) : SizeInt
```

Visibility: default

Description: Compares the null-terminated strings `S1` and `S2`. The result is

- A negative `Longint` when `S1<S2`.
- 0 when `S1=S2`.
- A positive `Longint` when `S1>S2`.

For an example, see StrLComp (1780).

Errors: None.

See also: StrLComp (1780), StrIComp (1777), StrLIComp (1782)

76.15.220 StrCopy

Synopsis: Copy a null-terminated string.

Declaration:
```
function strcopy(dest: PChar; source: PChar) : PChar;  Overload
function StrCopy(Dest: PWideChar; Source: PWideChar) : PWideChar
                ;  Overload
```

Visibility: default

Description: Copy the null terminated string in `Source` to `Dest`, and returns a pointer to `Dest`. `Dest` needs enough room to contain `Source`, i.e. `StrLen(Source)+1` bytes.

Errors: No length checking is performed.

See also: StrPCopy (1785), StrLCopy (1780), StrECopy (1775)

Listing: ./examples/stringex/ex4.pp

```
Program Example4;

Uses strings;

{ Program to demonstrate the StrCopy function. }

Const P : PCHar = 'This is a PCHAR string.';

var PP : PChar;

begin
  PP:=StrAlloc(Strlen(P)+1);
  STrCopy (PP,P);
  If StrComp (PP,P)<>0 then
    WriteIn ('Oh-oh problems...')
  else
    WriteIn ('All is well : PP=',PP);
  StrDispose(PP);
end.
```

76.15.221 StrDispose

Synopsis: Dispose of a null-terminated string on the heap.

Declaration:
```
procedure StrDispose(Str: PChar)
procedure StrDispose(str: PWideChar)
```

Visibility: default

Description: `StrDispose` frees any memory allocated for `Str`. This function will only function correctly if `Str` has been allocated on the heap, for example using StrAlloc (1772) or StrNew (1784) from the SysUtils unit.

For an example, see StrBufSize (1772).

Errors: If an invalid pointer is passed, or a pointer not allocated with `StrAlloc`, an error may occur.

See also: StrBufSize (1772), StrAlloc (1772), StrNew (1784)

76.15.222 strecopy

Synopsis: Copy a null-terminated string, return a pointer to the end.

Declaration:
```
function strecopy(dest: PChar; source: PChar) : PChar
function strecopy(dest: PWideChar; source: PWideChar) : PWideChar
```

Visibility: default

Description: Copies the Null-terminated string in `Source` to `Dest`, and returns a pointer to the end (i.e. the terminating Null-character) of the copied string.

Errors: No length checking is performed.

See also: StrLCopy (1780), StrCopy (1774)

Listing: ./examples/stringex/ex6.pp

```
Program Example6;

Uses strings;

{ Program to demonstrate the StrECopy function. }

Const P : PChar = 'This is a PCHAR string.';

Var PP : PChar;

begin
  PP:=StrAlloc (StrLen(P)+1);
  If Longint(StrECopy(PP,P))-Longint(PP)<>StrLen(P) then
    WriteIn('Something is wrong here !')
  else
    WriteIn ('PP= ',PP);
  StrDispose(PP);
end.
```

76.15.223 strend

Synopsis: Return a pointer to the end of a null-terminated string.

Declaration:
```
function strend(p: PChar) : PChar
function strend(p: PWideChar) : PWideChar
```

Visibility: default

Description: Returns a pointer to the end of `P`. (i.e. to the terminating null-character.

Errors: None.

See also: StrLen (1781)

Listing: ./examples/stringex/ex7.pp

```
Program Example6;

Uses strings;

{ Program to demonstrate the StrEnd function. }

Const P : PChar = 'This is a PCHAR string.';

begin
  If Longint(StrEnd(P))-Longint(P)<>StrLen(P) then
    WriteIn('Something is wrong here !')
  else
    WriteIn ('All is well..');
end.
```

76.15.224 StrFmt

Synopsis: Format a string with given arguments, store the result in a buffer.

Declaration:
```
function StrFmt(Buffer: PChar; Fmt: PChar; const args: Array of const)
                  : PChar
function StrFmt(Buffer: PChar; Fmt: PChar; const Args: Array of const;
                const FormatSettings: TFormatSettings) : PChar
```

Visibility: default

Description: `StrFmt` will format `fmt` with `Args`, as the Format (1735) function does, and it will store the result in `Buffer`. The function returns `Buffer`. `Buffer` should point to enough space to contain the whole result.

Errors: for a list of errors, see Format (1735).

See also: StrLFmt (1781), FmtStr (1734), Format (1735), FormatBuf (1742)

Listing: ./examples/sysutex/ex80.pp

```
Program Example80;

{ This program demonstrates the StrFmt function }

Uses sysutils;

Var S : AnsiString;

Begin
  SetLEngth(S,80);
  WriteIn (StrFmt (@S[1],'For some nice examples of fomatting see %s.',['Format']));
End.
```

76.15.225 stricomp

Synopsis: Compare 2 null-terminated strings, case insensitive.

Declaration:
```
function stricomp(str1: PChar; str2: PChar) : SizeInt
function stricomp(str1: PWideChar; str2: PWideChar) : SizeInt
```

Visibility: default

Description: Compares the null-terminated strings `S1` and `S2`, ignoring case. The result is

- A negative `Longint` when `S1<S2`.
- 0 when `S1=S2`.
- A positive `Longint` when `S1>S2`.

Errors: None.

See also: StrLComp (1780), StrComp (1774), StrLIComp (1782)

Listing: ./examples/stringex/ex8.pp

```
Program Example8;

Uses strings;

{ Program to demonstrate the StrLComp function. }

Const P1 : PChar = 'This is the first string.';
      P2 : PCHar = 'This is the second string.';

Var L : Longint;

begin
  Write ('P1 and P2 are ');
  If StrComp (P1,P2)<>0 then write ('NOT ');
  write ('equal. The first ');
  L:=1;
  While StrLComp(P1,P2,L)=0 do inc (L);
  dec(l);
  Writeln (l,' characters are the same.');
end.
```

76.15.226 StringOf

Synopsis: Create a Unicode string from an array of bytes.

Declaration:
```
function StringOf(const Bytes: TBytes) : UnicodeString
```

Visibility: default

Description: `StringOf` converts an array of bytes (`Bytes`) to a Unicode string. It interprets the bytes as a single-byte string, using the default codepage.

To create a string where the bytes are interpreted as wide chars, use WideStringOf (1818) instead.

Errors: None.

See also: WideBytesOf (1815), WideStringOf (1818)

76.15.227 StringReplace

Synopsis: Replace occurrences of one substring with another in a string.

Declaration:
```
function StringReplace(const S: string; const OldPattern: string;
                       const NewPattern: string; Flags: TReplaceFlags;
                       out aCount: Integer) : string
function StringReplace(const S: string; const OldPattern: string;
                       const NewPattern: string; Flags: TReplaceFlags)
                        : string
```

Visibility: default

Description: `StringReplace` searches the string `S` for occurrences of the string `OldPattern` and, if it is found, replaces it with `NewPattern`. It returns the resulting string. The behaviour of `StringReplace` can be tuned with `Flags`, which is of type TReplaceFlags (1659). Standard behaviour is to replace only the first occurrence of `OldPattern`, and to search case sensitively.

Errors: None.

See also: TReplaceFlags (1659), WideStringReplace (1818)

76.15.228 StringToGUID

Synopsis: Convert a string to a native `TGUID` type.

Declaration:
```
function StringToGUID(const S: string) : TGuid
```

Visibility: default

Description: `StringToGUID` converts the string `S` to a valid GUID. The string `S` should be of the form

```
{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}
```

Where each X is a hexadecimal digit. The dashes and braces are required.

Errors: In case `S` contains an invalid GUID representation, a EConvertError (1830) exception is raised.

See also: Supports (1798), #rtl.system.TGUID (1420), GUIDToString (1754), IsEqualGuid (1759)

76.15.229 strlcat

Synopsis: Concatenate 2 null-terminated strings, with length boundary.

Declaration:
```
function strlcat(dest: PChar; source: PChar; l: SizeInt) : PChar
function strlcat(dest: PWideChar; source: PWideChar; l: SizeInt)
                 : PWideChar
```

Visibility: default

Description: Adds `MaxLen` characters from `Source` to `Dest`, and adds a terminating null-character. Returns `Dest`.

Errors: None.

See also: StrCat (1773)

Listing: ./examples/stringex/ex12.pp

```
Program Example12;

Uses strings;

{ Program to demonstrate the StrLCat function. }

Const P1 : PChar = '1234567890';

Var P2 : PChar;

begin
  P2:=StrAlloc (StrLen(P1)*2+1);
  P2^:=#0; { Zero length }
  StrCat (P2,P1);
  StrLCat (P2,P1,5);
  WriteIn ('P2 = ',P2);
  StrDispose(P2)
end.
```

76.15.230 strlcomp

Synopsis: Compare limited number of characters of 2 null-terminated strings.

Declaration:
```
function strlcomp(str1: PChar; str2: PChar; l: SizeInt) : SizeInt
function strlcomp(str1: PWideChar; str2: PWideChar; l: SizeInt)
                  : SizeInt
```

Visibility: default

Description: Compares maximum `L` characters of the null-terminated strings `S1` and `S2`. The result is

- A negative `Longint` when `S1<S2`.
- 0 when `S1=S2`.
- A positive `Longint` when `S1>S2`.

Errors: None.

See also: StrComp (1774), StrIComp (1777), StrLIComp (1782)

Listing: ./examples/stringex/ex8.pp

```
Program Example8;

Uses strings;

{ Program to demonstrate the StrLComp function. }

Const P1 : PChar = 'This is the first string.';
      P2 : PCHar = 'This is the second string.';

Var L : Longint;

begin
  Write ('P1 and P2 are ');
  If StrComp (P1,P2)<>0 then write ('NOT ');
  write ('equal. The first ');
  L:=1;
  While StrLComp(P1,P2,L)=0 do inc (L);
  dec(l);
  Writeln (l,' characters are the same.');
end.
```

76.15.231 StrLCopy

Synopsis: Copy a null-terminated string, limited in length.

Declaration:
```
function strlcopy(dest: PChar; source: PChar; maxlen: SizeInt) : PChar
                  ;  Overload
function StrLCopy(Dest: PWideChar; Source: PWideChar; MaxLen: SizeInt)
                  : PWideChar;  Overload
```

Visibility: default

Description: Copies `MaxLen` characters from `Source` to `Dest`, and makes `Dest` a null terminated string.

Errors: No length checking is performed.

See also: StrCopy (1774), StrECopy (1775)

Listing: ./examples/stringex/ex5.pp

```
Program Example5;

Uses strings;

{ Program to demonstrate the StrLCopy function. }

Const P : PCHar = '123456789ABCDEF';

var PP : PCHar;

begin
  PP:=StrAlloc(11);
  WriteIn ('First 10 characters of P : ',StrLCopy (PP,P,10));
  StrDispose(PP);
end.
```

76.15.232 StrLen

Synopsis: Length of a null-terminated string.

Declaration:
```
function strlen(p: PChar) : SizeInt;  Overload
function StrLen(p: PWideChar) : SizeInt;  Overload
```

Visibility: default

Description: Returns the length of the null-terminated string `P`. If `P` equals `Nil` then zero (0) is returned.

Errors: None.

See also: StrNew (1784)

Listing: ./examples/stringex/ex1.pp

```
Program Example1;

Uses strings;

{ Program to demonstrate the StrLen function. }

Const P : PChar = 'This is a constant pchar string';

begin
  WriteIn ('P         : ',p);
  WriteIn ('length(P) : ',StrLen(P));
end.
```

76.15.233 StrLFmt

Synopsis: Format a string with given arguments, but with limited length.

Declaration:
```
function StrLFmt(Buffer: PChar; Maxlen: Cardinal; Fmt: PChar;
                 const args: Array of const) : PChar
function StrLFmt(Buffer: PChar; Maxlen: Cardinal; Fmt: PChar;
                 const args: Array of const;
                 const FormatSettings: TFormatSettings) : PChar
```

Visibility: default

Description: `StrLFmt` will format `fmt` with `Args`, as the Format (1735) function does, and it will store maximally `Maxlen characters` of the result in `Buffer`. The function returns `Buffer`. `Buffer` should point to enough space to contain `MaxLen` characters.

Errors: for a list of errors, see Format (1735).

See also: StrFmt (1777), FmtStr (1734), Format (1735), FormatBuf (1742)

Listing: ./examples/sysutex/ex81.pp

```
Program Example80;

{ This program demonstrates the StrFmt function }

Uses sysutils;

Var S : AnsiString;

Begin
  SetLEngth(S,80);
  Writeln (StrLFmt (@S[1],80,'For some nice examples of fomatting see %s.',['Format']));
End.
```

76.15.234 strlicomp

Synopsis: Compare limited number of characters in 2 null-terminated strings, ignoring case.

Declaration:
```
function strlicomp(str1: PChar; str2: PChar; l: SizeInt) : SizeInt
function strlicomp(str1: PWideChar; str2: PWideChar; l: SizeInt)
                   : SizeInt
```

Visibility: default

Description: Compares maximum `L` characters of the null-terminated strings `S1` and `S2`, ignoring case. The result is

- A negative `Longint` when `S1<S2`.
- 0 when `S1=S2`.
- A positive `Longint` when `S1>S2`.

For an example, see StrIComp (1777)

Errors: None.

See also: StrLComp (1780), StrComp (1774), StrIComp (1777)

76.15.235 strlower

Synopsis: Convert null-terminated string to all-lowercase.

Declaration:
```
function strlower(p: PChar) : PChar
function strlower(p: PWideChar) : PWideChar
```

Visibility: default

Description: Converts `P` to an all-lowercase string. Returns `P`.

Errors: None.

See also: StrUpper (1798)

Listing: ./examples/stringex/ex14.pp

```
Program Example14;

Uses strings;

{ Program to demonstrate the StrLower and StrUpper functions. }

Const
    P1 : PChar = 'THIS IS AN UPPERCASE PCHAR STRING';
    P2 : PChar = 'this is a lowercase string';


begin
  P1:=StrNew(P1);
  P2:=strNew(P2);
  WriteIn ('Uppercase : ',StrUpper(P2));
  StrLower(P1);
  WriteIn ('Lowercase : ',P1);
  StrDispose(P1);
  StrDispose(P2);
end.
```

76.15.236 StrMove

Synopsis: Move a null-terminated string to new location.

Declaration:
```
function strmove(dest: PChar; source: PChar; l: SizeInt) : PChar
                 ;  Overload
function StrMove(dest: PWideChar; source: PWideChar; l: SizeInt)
                  : PWideChar;  Overload
```

Visibility: default

Description: Copies `MaxLen` characters from `Source` to `Dest`. No terminating null-character is copied. Returns `Dest`

Errors: None.

See also: StrLCopy (1780), StrCopy (1774)

Listing: ./examples/stringex/ex10.pp

```
Program Example10;

Uses strings;

{ Program to demonstrate the StrMove function. }

Const P1 : PCHAR = 'This is a pchar string.';


Var P2 : Pchar;

begin
  P2:=StrAlloc(StrLen(P1)+1);
  StrMove (P2,P1,StrLen(P1)+1); { P2:=P1 }
  Writeln ('P2 = ',P2);
  StrDispose(P2);
end.
```

76.15.237 strnew

Synopsis: Allocate room for new null-terminated string.

Declaration:
```
function strnew(p: PChar) : PChar;  Overload
function strnew(p: PWideChar) : PWideChar;  Overload
```

Visibility: default

Description: Copies `P` to the Heap, and returns a pointer to the copy.

Errors: Returns `Nil` if no memory was available for the copy.

See also: StrCopy (1774), StrDispose (1775)

Listing: ./examples/stringex/ex16.pp

```
Program Example16;

Uses strings;

{ Program to demonstrate the StrNew function. }

Const P1 : PChar = 'This is a PChar string';

var P2 : PChar;

begin
  P2:=StrNew (P1);
  If P1=P2 then
    writeln ('This can''t be happening...')
  else
    writeln ('P2 : ',P2);
  StrDispose(P2);
end.
```

76.15.238 StrNextChar

Synopsis: Returns a pointer to the location of the next empty character in a null-terminated string.

Declaration:
```
function StrNextChar(const Str: PChar) : PChar
```

Visibility: default

Description: `StrNextChar` returns a pointer to the null-character that terminates the string `Str`

Errors: if `Str` is not properly terminated, an access violation may occur.

76.15.239 StrPas

Synopsis: Convert a null-terminated string to an ansistring.

Declaration:
```
function StrPas(Str: PChar) : string;  Overload
function StrPas(Str: PWideChar) : UnicodeString;  Overload
```

Visibility: default

Description: Converts a null terminated string in `Str` to an Ansitring, and returns this string. This string is NOT truncated at 255 characters as is the system unit's version.

Errors: None.

See also: StrPCopy (1785), StrPLCopy (1785)

76.15.240 StrPCopy

Synopsis: Copy an ansistring to a null-terminated string.

Declaration:
```
function StrPCopy(Dest: PChar; const Source: string) : PChar;  Overload
function StrPCopy(Dest: PWideChar; const Source: UnicodeString)
                  : PWideChar;  Overload
```

Visibility: default

Description: `StrPCopy` Converts the Ansistring in `Source` to a Null-terminated string, and copies it to `Dest`. `Dest` needs enough room to contain the string `Source`, i.e. `Length(Source)+1` bytes.

Errors: No checking is performed to see whether `Dest` points to enough memory to contain `Source`.

See also: StrPLCopy (1785), StrPas (1785)

76.15.241 StrPLCopy

Synopsis: Copy a limited number of characters from an ansistring to a null-terminated string.

Declaration:
```
function StrPLCopy(Dest: PChar; const Source: string; MaxLen: SizeUInt)
                   : PChar;  Overload
function StrPLCopy(Dest: PWideChar; const Source: UnicodeString;
                   MaxLen: SizeUInt) : PWideChar;  Overload
```

Visibility: default

Description: `StrPLCopy` Converts maximally `MaxLen` characters of the Ansistring in `Source` to a Null-terminated string, and copies it to `Dest`. `Dest` needs enough room to contain the characters.

Errors: No checking is performed to see whether `Dest` points to enough memory to contain L characters of `Source`.

See also: StrPCopy (1785)

76.15.242 strpos

Synopsis: Find position of one null-terminated substring in another.

Declaration:
```
function strpos(str1: PChar; str2: PChar) : PChar
function strpos(str1: PWideChar; str2: PWideChar) : PWideChar
```

Visibility: default

Description: Returns a pointer to the first occurrence of `S2` in `S1`. If `S2` does not occur in `S1`, returns `Nil`.

Errors: None.

See also: StrScan (1787), StrRScan (1786)

Listing: ./examples/stringex/ex15.pp

```
Program Example15;

Uses strings;

{ Program to demonstrate the StrPos function. }

Const P : PChar = 'This is a PChar string.';
      S : Pchar = 'is';
begin
  WriteIn ('Position of ''is'' in P : ',sizeint(StrPos(P,S))-sizeint(P));
end.
```

76.15.243 strrscan

Synopsis: Find last occurrence of a character in a null-terminated string.

Declaration:
```
function strrscan(p: PChar; c: Char) : PChar
function strrscan(p: PWideChar; c: WideChar) : PWideChar
```

Visibility: default

Description: Returns a pointer to the last occurrence of the character `C` in the null-terminated string `P`. If `C` does not occur, returns `Nil`.

For an example, see StrScan (1787).

Errors: None.

See also: StrScan (1787), StrPos (1786)

76.15.244 StrScan

Synopsis: Find first occurrence of a character in a null-terminated string.

Declaration:
```
function strscan(p: PChar; c: Char) : PChar;  Overload
function StrScan(P: PWideChar; C: WideChar) : PWideChar;  Overload
```

Visibility: default

Description: Returns a pointer to the first occurrence of the character `C` in the null-terminated string `P`. If `C` does not occur, returns `Nil`.

Errors: None.

See also: StrRScan (1786), StrPos (1786)

Listing: ./examples/stringex/ex13.pp

```
Program Example13;

Uses strings;

{ Program to demonstrate the StrScan and StrRScan functions. }

Const P : PChar = 'This is a PCHAR string.';
      S : Char = 's' ;

begin
  Writeln ('P, starting from first ''s'' : ',StrScan(P,s));
  Writeln ('P, starting from last ''s'' : ',StrRScan(P,s));
end.
```

76.15.245 StrToBool

Synopsis: Convert a string to a boolean value.

Declaration:
```
function StrToBool(const S: string) : Boolean
function StrToBool(const S: string;
                   const FormatSettings: TFormatSettings) : Boolean
```

Visibility: default

Description: `StrToBool` will convert the string `S` to a boolean value. The string `S` can contain one of `'True'`, `'False'` (case is ignored) or a numerical value. If it contains a numerical value, 0 is converted to `False`, all other values result in `True`. If the string `S` contains no valid boolean, then an EConvertError (1830) exception is raised.

Errors: On error, an EConvertError (1830) exception is raised.

See also: BoolToStr (1685)

76.15.246 StrToBoolDef

Synopsis: Convert string to boolean value, returning default in case of error.

Declaration:
```
function StrToBoolDef(const S: string; Default: Boolean) : Boolean
function StrToBoolDef(const S: string; Default: Boolean;
                      const FormatSettings: TFormatSettings) : Boolean
```

Visibility: default

Description: `StrToBoolDef` tries to convert the string `S` to a boolean value, and returns the boolean value in case of success. In case `S` does not contain a valid boolean string, `Default` is returned.

See also: StrToBool (1787), TryStrToBool (1806)

76.15.247 StrToCurr

Synopsis: Convert a string to a currency value.

Declaration:
```
function StrToCurr(const S: string) : Currency
function StrToCurr(const S: string;
                   const FormatSettings: TFormatSettings) : Currency
```

Visibility: default

Description: `StrToCurr` converts a string to a currency value and returns the value. The string should contain a valid currency amount, without currency symbol. If the conversion fails, an EConvertError (1830) exception is raised.

Errors: On error, an EConvertError (1830) exception is raised.

See also: CurrToStr (1693), StrToCurrDef (1788)

76.15.248 StrToCurrDef

Synopsis: Convert a string to a currency value, using a default value.

Declaration:
```
function StrToCurrDef(const S: string; Default: Currency) : Currency
function StrToCurrDef(const S: string; Default: Currency;
                      const FormatSettings: TFormatSettings) : Currency
```

Visibility: default

Description: `StrToCurrDef` converts a string to a currency value and returns the value. The string should contain a valid currency amount, without currency symbol. If the conversion fails, the fallback `Default` value is returned.

Errors: On error, the `Default` value is returned.

See also: CurrToStr (1693), StrToCurr (1788)

76.15.249 StrToDate

Synopsis: Convert a date string to a `TDateTime` value.

Declaration:
```
function StrToDate(const S: ShortString) : TDateTime
function StrToDate(const S: Ansistring) : TDateTime
function StrToDate(const S: ShortString; separator: Char) : TDateTime
function StrToDate(const S: AnsiString; separator: Char) : TDateTime
function StrToDate(const S: string; FormatSettings: TFormatSettings)
                    : TDateTime
function StrToDate(const S: ShortString; const useformat: string;
                   separator: Char) : TDateTime
function StrToDate(const S: AnsiString; const useformat: string;
                   separator: Char) : TDateTime
```

```
function StrToDate(const S: PChar; Len: Integer;
                   const useformat: string; separator: Char) : TDateTime
```

Visibility: default

Description: `StrToDate` converts the string `S` to a `TDateTime` date value. The Date must consist of 1 to three digits, separated by the `DateSeparator` character. If two numbers are given, they are supposed to form the day and month of the current year. If only one number is given, it is supposed to represent the day of the current month. (This is *not* supported in Delphi)

The order of the digits (y/m/d, m/d/y, d/m/y) is determined from the `ShortDateFormat` variable.

Errors: On error (e.g. an invalid date or invalid character), an EConvertError (1830) exception is raised.

See also: StrToTime (1795), DateToStr (1698), TimeToStr (1802), EConvertError (1830)

Listing: ./examples/sysutex/ex19.pp

```
Program Example19;

{ This program demonstrates the StrToDate function }

Uses sysutils;

Procedure TestStr (S : String);

begin
  WriteIn (S,' : ',DateToStr(StrToDate(S)));
end;

Begin

  WriteIn ('ShortDateFormat ',ShortDateFormat);
  TestStr(DateTimeToStr(Date));
  TestStr('05'+DateSeparator+'05'+DateSeparator+'1999');
  TestStr('5'+DateSeparator+'5');
  TestStr('5');
End.
```

76.15.250 StrToDateDef

Synopsis: Convert string to date, returning a default value.

Declaration:
```
function StrToDateDef(const S: ShortString; const Defvalue: TDateTime)
                      : TDateTime
function StrToDateDef(const S: ShortString; const Defvalue: TDateTime;
                      separator: Char) : TDateTime
function StrToDateDef(const S: AnsiString; const Defvalue: TDateTime)
                      : TDateTime
function StrToDateDef(const S: AnsiString; const Defvalue: TDateTime;
                      separator: Char) : TDateTime
```

Visibility: default

Description: `StrToDateDef` tries to convert the string `S` to a valid `TDateTime` date value, and returns `DefValue` if `S` does not contain a valid date indication.

Errors: None.

See also: StrToDate (1788), TryStrToDate (1807), StrToTimeDef (1796)

76.15.251 StrToDateTime

Synopsis: Convert a date/time string to a `TDateTime` value.

Declaration:
```
function StrToDateTime(const S: AnsiString) : TDateTime
function StrToDateTime(const s: ShortString;
                       const FormatSettings: TFormatSettings) : TDateTime
function StrToDateTime(const s: AnsiString;
                       const FormatSettings: TFormatSettings) : TDateTime
```

Visibility: default

Description: `StrToDateTime` converts the string `S` to a `TDateTime` date and time value. The date and time parts must be separated by a space.

For the date part, the same restrictions apply as for the StrToDate (1788) function: The Date must consist of 1 to three numbers, separated by the `DateSeparator` character. If two numbers are given, they are supposed to form the day and month of the current year. If only one number is given, it is supposed to represent the day of the current month. (This is *not* supported in Delphi)

The order of the 3 numbers (y/m/d, m/d/y, d/m/y) is determined from the `ShortDateFormat` variable.

Errors: On error (e.g. an invalid date or invalid character), an EConvertError (1830) exception is raised.

See also: StrToDate (1788), StrToTime (1795), DateTimeToStr (1695), EConvertError (1830)

Listing: ./examples/sysutex/ex20.pp

```
Program Example20;

{ This program demonstrates the StrToDateTime function }

Uses sysutils;

Procedure TestStr (S : String);

begin
  WriteIn (S,' : ',DateTimeToStr(StrToDateTime(S)));
end;

Begin

  WriteIn ('ShortDateFormat ',ShortDateFormat);
  TestStr(DateTimeToStr(Now));
  TestStr('05-05-1999 15:50');
  TestStr('5-5 13:30');
  TestStr('5 1:30PM');
End.
```

76.15.252 StrToDateTimeDef

Synopsis: Convert string to date/time, returning a default value.

Declaration:
```
function StrToDateTimeDef(const S: ShortString;
                          const Defvalue: TDateTime) : TDateTime
function StrToDateTimeDef(const S: AnsiString;
                          const Defvalue: TDateTime) : TDateTime
```

```
function StrToDateTimeDef(const S: AnsiString;
                          const Defvalue: TDateTime;
                          const FormatSettings: TFormatSettings)
                           : TDateTime
```

Visibility: default

Description: `StrToDateTimeDef` tries to convert the string `S` to a valid `TDateTime` date and time value, and returns `DefValue` if `S` does not contain a valid date-time indication.

Errors: None.

See also: StrToTimeDef (1796), StrToDateDef (1789), TryStrToDateTime (1807), StrToDateTime (1790)

76.15.253 StrToDWord

Synopsis: Convert string to DWord (cardinal).

Declaration: `function StrToDWord(const s: string) : DWord`

Visibility: default

Description: `StrToDWord` will convert the string `S` to a `DWord` value, and returns the value.

Errors: In case the string `S` is not a valid number, or is a value outside the `DWord` range, an `EConvertError` exception will be raised.

See also: TryStrToDWord (1808), StrToDWordDef (1791)

76.15.254 StrToDWordDef

Synopsis: Convert string to DWord (cardinal), using default.

Declaration: `function StrToDWordDef(const S: string; Default: DWord) : DWord`

Visibility: default

Description: `StrToDWordDef` will try to convert the string `S` to a `DWord` value. If the conversion was successful, it returns the value.

If the conversion failed, the fallback value in `Default` is returned.

See also: TryStrToDWord (1808), StrToDWord (1791)

76.15.255 StrToFloat

Synopsis: Convert a string to a floating-point value.

Declaration:
```
function StrToFloat(const S: string) : Extended
function StrToFloat(const S: string;
                    const FormatSettings: TFormatSettings) : Extended
```

Visibility: default

Description: `StrToFloat` converts the string `S` to a floating point value. `S` should contain a valid string representation of a floating point value (either in decimal or scientific notation). The `thousandseparator` character may however not be used.

Up to and including version 2.2.2 of the compiler, if the string contains a decimal value, then the decimal separator character can either be a '.' or the value of the `DecimalSeparator` variable.

As of version 2.3.1, the string may contain only the `DecimalSeparator` character. The dot ('.') can no longer be used instead of the `DecimalSeparator`.

Errors: If the string `S` doesn't contain a valid floating point string, then an exception will be raised.

See also: TextToFloat (1800), FloatToStr (1729), FormatFloat (1743), StrToInt (1793)

Listing: ./examples/sysutex/ex90.pp

```
Program Example90;

{ This program demonstrates the StrToFloat function }
{$mode objfpc}
{$h+ }

Uses SysUtils;

Const
  NrValues = 5;
  TestStr : Array[1..NrValues] of string =
            ('1,1','-0,2','1,2E-4','0','1E4');

Procedure Testit;

Var
  I : Integer;
  E : Extended;

begin
  Writeln('Using DecimalSeparator : ',DecimalSeparator);
  For I:=1 to NrValues do
    begin
    Writeln('Converting : ',TestStr[i]);
    Try
      E:=StrToFloat(TestStr[i]);
      Writeln('Converted value : ',E);
    except
      On E : Exception do
        Writeln('Exception when converting : ',E.Message);
    end;
    end;
end;

Begin
  DecimalSeparator:=',';
  Testit;
  DecimalSeparator:='.';
  Testit;
End.
```

76.15.256 StrToFloatDef

Synopsis: Convert a string to a float, with a default value.

Declaration:
```
function StrToFloatDef(const S: string; const Default: Extended)
                            : Extended
function StrToFloatDef(const S: string; const Default: Extended;
                       const FormatSettings: TFormatSettings) : Extended
```

Visibility: default

Description: `StrToFloatDef` tries to convert the string `S` to a floating point value, and returns this value. If the conversion fails for some reason, the value `Default` is returned instead.

Errors: None. On error, the `Default` value is returned.

76.15.257 StrToInt

Synopsis: Convert a string to an integer value.

Declaration:
```
function StrToInt(const s: string) : LongInt
```

Visibility: default

Description: `StrToInt` will convert the string `S` to a 32-bit signed integer. If the string contains invalid characters or has an invalid format, then an EConvertError (1830) is raised.

To be successfully converted, a string can contain a combination of `numerical` characters, possibly preceded by a minus sign (-). Spaces and tab characters are only allowed at the start of the string. Internally, Val (1598) is used, the rules of `Val` apply.

The string `S` can contain a number in decimal, hexadecimal, binary or octal format, as described in the language reference. For enumerated values, the string must be the name of the enumerated value. The name is searched case insensitively.

For hexadecimal values, the prefix '0x' or 'x' (case insensitive) may be used as well.

Errors: In case of error, an `EConvertError` is raised.

See also: Val (1598), IntToStr (1758), StrToUint (1797), StrToInt64 (1794), StrToIntDef (1794), EConvertError (1830)

Listing: ./examples/sysutex/ex82.pp

```
Program Example82;

{$mode objfpc}

{ This program demonstrates the StrToInt function }

Uses sysutils;

Begin
  WriteLn (StrToInt('1234'));
  WriteLn (StrToInt('-1234'));
  WriteLn (StrToInt('0'));
  Try
    WriteLn (StrToInt('12345678901234567890'));
  except
    On E : EConvertError do
      WriteLn ('Invalid number encountered');
```

```
  end;
End.
```

76.15.258 StrToInt64

Synopsis: Convert a string to an Int64 value.

Declaration: `function StrToInt64(const s: string) : Int64`

Visibility: default

Description: `StrToInt64` converts the string `S` to a Int64 value, and returns this value. The string can only contain numerical characters, and optionally a minus sign as the first character. Whitespace is not allowed.

Hexadecimal values (starting with the $ character) are supported.

Errors: On error, a EConvertError (1830) exception is raised.

See also: TryStrToInt64 (1809), StrToInt64Def (1794), StrToInt (1793), TryStrToInt (1808), StrToIntDef (1794)

76.15.259 StrToInt64Def

Synopsis: Convert a string to an Int64 value, with a default value.

Declaration: `function StrToInt64Def(const S: string; Default: Int64) : Int64`

Visibility: default

Description: `StrToInt64Def` tries to convert the string `S` to a Int64 value, and returns this value. If the conversion fails for some reason, the value `Default` is returned instead.

Errors: None. On error, the `Default` value is returned.

See also: StrToInt64 (1794), TryStrToInt64 (1809), StrToInt (1793), TryStrToInt (1808), StrToIntDef (1794)

76.15.260 StrToIntDef

Synopsis: Convert a string to an integer value, with a default value.

Declaration: `function StrToIntDef(const S: string; Default: LongInt) : LongInt`

Visibility: default

Description: `StrToIntDef` will convert a string to an integer. If the string contains invalid characters or has an invalid format, then `Default` is returned.

To be successfully converted, a string can contain a combination of `numerical` characters, possibly preceded by a minus sign (-). Spaces are not allowed.

Errors: None.

See also: IntToStr (1758), StrToInt (1793)

Listing: ./examples/sysutex/ex83.pp

```
Program Example82;

{$mode objfpc}

{ This program demonstrates the StrToInt function }

Uses sysutils;

Begin
  WriteLn (StrToIntDef('1234',0));
  WriteLn (StrToIntDef('-1234',0));
  WriteLn (StrToIntDef('0',0));
  Try
    WriteLn (StrToIntDef('12345678901234567890',0));
  except
    On E : EConvertError do
      WriteLn ('Invalid number encountered');
  end;
End.
```

76.15.261 StrToQWord

Synopsis: Convert a string to a QWord.

Declaration: `function StrToQWord(const s: string) : QWord`

Visibility: default

Description: `TryStrToQWord` converts the string `S` to a valid QWord (unsigned 64-bit) value, and returns the result.

Errors: If the string `S` does not contain a valid QWord value, a EConvertError (1830) exception is raised.

See also: TryStrToQWord (1809), StrToQWordDef (1795), StrToInt64 (1794), StrToInt (1793)

76.15.262 StrToQWordDef

Synopsis: Try to convert a string to a QWord, returning a default value in case of failure.

Declaration: `function StrToQWordDef(const S: string; Default: QWord) : QWord`

Visibility: default

Description: `StrToQWordDef` tries to convert the string `S` to a valid QWord (unsigned 64-bit) value, and returns the result. If the conversion fails, the function returns the value passed in `Def`.

See also: StrToQWord (1795), TryStrToQWord (1809), StrToInt64Def (1794), StrToIntDef (1794)

76.15.263 StrToTime

Synopsis: Convert a time string to a `TDateTime` value.

Declaration:
```
function StrToTime(const S: Shortstring) : TDateTime
function StrToTime(const S: Ansistring) : TDateTime
function StrToTime(const S: ShortString; separator: Char) : TDateTime
function StrToTime(const S: AnsiString; separator: Char) : TDateTime
```

```
function StrToTime(const S: string; FormatSettings: TFormatSettings)
                   : TDateTime
function StrToTime(const S: PChar; Len: Integer; separator: Char)
                   : TDateTime
```

Visibility: default

Description: `StrToTime` converts the string `S` to a `TDateTime` time value. The time must consist of 1 to 4 digits, separated by the `TimeSeparator` character. If two numbers are given, they are supposed to form the hour and minutes.

Errors: On error (e.g. an invalid date or invalid character), an EConvertError (1830) exception is raised.

See also: StrToDate (1788), StrToDateTime (1790), TimeToStr (1802), EConvertError (1830)

Listing: ./examples/sysutex/ex21.pp

```
Program Example21;

{ This program demonstrates the StrToTime function }

Uses sysutils;

Procedure TestStr (S : String);

begin
  WriteIn (S,' : ',TimeToStr(StrToTime(S)));
end;

Begin
  teststr (TimeToStr(Time));
  teststr ('12:00');
  teststr ('15:30');
  teststr ('3:30PM');
End.
```

76.15.264 StrToTimeDef

Synopsis: Convert string to time, returning a default value.

Declaration:
```
function StrToTimeDef(const S: ShortString; const Defvalue: TDateTime)
                      : TDateTime
function StrToTimeDef(const S: ShortString; const Defvalue: TDateTime;
                      separator: Char) : TDateTime
function StrToTimeDef(const S: AnsiString; const Defvalue: TDateTime)
                      : TDateTime
function StrToTimeDef(const S: AnsiString; const Defvalue: TDateTime;
                      separator: Char) : TDateTime
```

Visibility: default

Description: `StrToTimeDef` tries to convert the string `S` to a valid `TDateTime` time value, and returns `DefValue` if `S` does not contain a valid time indication.

Errors: None.

See also: StrToTime (1795), TryStrToTime (1809), StrToDateDef (1789)

76.15.265 StrToUInt

Synopsis: Convert a string to unsigned integer value.

Declaration: `function StrToUInt(const s: string) : Cardinal`

Visibility: default

Description: `StrToUInt` will convert the string `S`to an unsigned 32-bit integer. If the string contains invalid characters or has an invalid format, then an EConvertError (1830) is raised.

To be successfully converted, a string can contain a combination of `numerical` characters. Spaces are not allowed.

The string `S` can contain a number in decimal, hexadecimal, binary or octal format, as described in the language reference.

For hexadecimal values, the prefix '0x' or '$x' (case insensitive) may be used as well.

Errors: In case of error, an `EConvertError` is raised.

See also: UIntToStr (1810), StrToInt (1793), StrToInt64 (1794), StrToUInt64 (1797), StrToIntDef (1794), EConvertError (1830)

76.15.266 StrToUInt64

Synopsis: Convert a string to unsigned integer value.

Declaration: `function StrToUInt64(const s: string) : UInt64`

Visibility: default

Description: `StrToUInt64` will convert the string `S`to an unsigned 64-bit integer. If the string contains invalid characters or has an invalid format, then an EConvertError (1830) is raised.

To be successfully converted, a string can contain a combination of `numerical` characters, possibly preceded by a minus sign (-). Spaces are not allowed.

The string `S` can contain a number in decimal, hexadecimal, binary or octal format, as described in the language reference.

For hexadecimal values, the prefix '0x' or '$x' (case insensitive) may be used as well.

Errors: In case of error, an `EConvertError` is raised.

See also: UIntToStr (1810), StrToInt (1793), StrToInt64 (1794), StrToUInt64 (1797), StrToIntDef (1794), EConvertError (1830)

76.15.267 StrToUInt64Def

Synopsis: Convert a string to an unsigned 64-bit integer value, with a default value.

Declaration: `function StrToUInt64Def(const S: string; Default: UInt64) : UInt64`

Visibility: default

Description: `StrToIntDef` will convert a string to an unsigned 64-bit integer. If the string contains invalid characters or has an invalid format, then `Default` is returned.

To be successfully converted, a string can contain a combination of `numerical` characters. Spaces are not allowed.

Errors: None.

See also: IntToStr (1758), StrToUInt64 (1797), TryStrToUInt64 (1810), StrToInt64 (1794)

76.15.268 StrToUIntDef

Synopsis: Convert a string to an unsigned integer value, with a default value.

Declaration:
```
function StrToUIntDef(const S: string; Default: Cardinal) : Cardinal
```

Visibility: default

Description: `StrToIntDef` will convert a string to an integer. If the string contains invalid characters or has an invalid format, then `Default` is returned.

To be successfully converted, a string can contain a combination of `numerical` characters. Spaces are not allowed.

Errors: None.

See also: IntToStr (1758), StrToUInt (1797), TryStrToUInt (1810), StrToInt (1793)

76.15.269 strupper

Synopsis: Convert null-terminated string to all-uppercase.

Declaration:
```
function strupper(p: PChar) : PChar
function strupper(p: PWideChar) : PWideChar
```

Visibility: default

Description: Converts `P` to an all-uppercase string. Returns `P`.

For an example, see StrLower (1783)

Errors: None.

See also: StrLower (1783)

76.15.270 Supports

Synopsis: Check whether a class or given interface supports an interface.

Declaration:
```
function Supports(const Instance: IInterface; const AClass: TClass;
                  out Obj) : Boolean;  Overload
function Supports(const Instance: IInterface; const IID: TGuid;
                  out Intf) : Boolean;  Overload
function Supports(const Instance: TObject; const IID: TGuid; out Intf)
                   : Boolean;  Overload
function Supports(const Instance: TObject; const IID: Shortstring;
                  out Intf) : Boolean;  Overload
function Supports(const Instance: IInterface; const AClass: TClass)
                   : Boolean;  Overload
function Supports(const Instance: IInterface; const IID: TGuid)
                   : Boolean;  Overload
function Supports(const Instance: TObject; const IID: TGuid) : Boolean
                  ;  Overload
function Supports(const Instance: TObject; const IID: Shortstring)
                   : Boolean;  Overload
function Supports(const AClass: TClass; const IID: TGuid) : Boolean
                  ;  Overload
function Supports(const AClass: TClass; const IID: Shortstring)
                   : Boolean;  Overload
```

Visibility: default

Description: `Supports` checks whether `Instance` supports the interface identified by `IID`. It returns `True` if it is supported, `False`. Optionally, a pointer to the interface is returned to `Intf`.

Errors: None.

See also: StringToGUID (1779)

76.15.271 SysErrorMessage

Synopsis: Format a system error message.

Declaration: `function SysErrorMessage(ErrorCode: Integer) : string`

Visibility: default

Description: `SysErrorMessage` returns a string that describes the operating system error code `ErrorCode`.

Errors: This routine may not be implemented on all platforms.

See also: EOSError (1836)

76.15.272 SystemTimeToDateTime

Synopsis: Convert a system time to a `TDateTime` value.

Declaration: `function SystemTimeToDateTime(const SystemTime: TSystemTime) : TDateTime`

Visibility: default

Description: `SystemTimeToDateTime` converts a `TSystemTime` record to a `TDateTime` style date/time indication.

Errors: None.

See also: DateTimeToSystemTime (1697)

Listing: ./examples/sysutex/ex22.pp

```
Program Example22;

{ This program demonstrates the SystemTimeToDateTime function }

Uses sysutils;

Var ST : TSystemTime;

Begin
  DateTimeToSystemTime(Now,ST);
  With St do
    begin
    WriteIn ('Today is     ',year,'/',month,'/',Day);
    WriteIn ('The time is ',Hour,':',minute,':',Second,'.',MilliSecond);
    end;
  WriteIn ('Converted : ',DateTimeToStr(SystemTimeToDateTime(ST)));
End.
```

76.15.273 TextToFloat

Synopsis: Convert a buffer to a float value.

Declaration:
```
function TextToFloat(Buffer: PChar; out Value: Extended) : Boolean
function TextToFloat(Buffer: PChar; out Value: Extended;
                     const FormatSettings: TFormatSettings) : Boolean
function TextToFloat(Buffer: PChar; out Value; ValueType: TFloatValue)
                      : Boolean
function TextToFloat(Buffer: PChar; out Value; ValueType: TFloatValue;
                     const FormatSettings: TFormatSettings) : Boolean
```

Visibility: default

Description: `TextToFloat` converts the string in `Buffer` to a floating point value. `Buffer` should contain a valid string representation of a floating point value (either in decimal or scientific notation).

If the buffer contains a decimal value, then the decimal separator character must be the value of the `DecimalSeparator` variable.

Remark Note that this behaviour has changed, earlier implementations also allowed the use of '.' in addition to the decimal separator character.

The function returns `True` if the conversion was successful.

Errors: If there is an invalid character in the buffer, then the function returns `False`

See also: StrToFloat (1791), FloatToStr (1729), FormatFloat (1743)

Listing: ./examples/sysutex/ex91.pp

```
Program Example91;

{ This program demonstrates the TextToFloat function }
{$mode objfpc}
{$h+ }

Uses SysUtils;

Const
  NrValues = 5;
  TestStr : Array[1..NrValues] of pchar =
          ('1,1','-0,2','1,2E-4','0','1E4');

Procedure Testit;

Var
  I : Integer;
  E : Extended;

begin
  Writeln('Using DecimalSeparator : ',DecimalSeparator);
  For I:=1 to NrValues do
    begin
    Writeln('Converting : ',TestStr[i]);
    If TextToFloat(TestStr[i],E) then
      Writeln('Converted value : ',E)
    else
      Writeln('Unable to convert value.');
    end;
end;
```

```
Begin
  DecimalSeparator:=',';
  Testit;
  DecimalSeparator:='.';
  Testit;
End.
```

76.15.274 Time

Synopsis: Returns the current time.

Declaration: `function Time : TDateTime`

Visibility: default

Description: `Time` returns the current time in `TDateTime` format. The date part of the `TDateTimeValue` is set to zero.

Errors: None.

See also: Now (1765), Date (1694)

Listing: ./examples/sysutex/ex23.pp

```
Program Example23;

{ This program demonstrates the Time function }

Uses sysutils;

Begin
  WriteIn ('The time is : ',TimeToStr(Time));
End.
```

76.15.275 TimeStampToDateTime

Synopsis: Convert a TimeStamp value to a `TDateTime` value.

Declaration: `function TimeStampToDateTime(const TimeStamp: TTimeStamp) : TDateTime`

Visibility: default

Description: `TimeStampToDateTime` converts `TimeStamp` to a `TDateTime` format variable. It is the inverse operation of DateTimeToTimeStamp (1697).

Errors: None.

See also: DateTimeToTimeStamp (1697), TimeStampToMSecs (1802)

Listing: ./examples/sysutex/ex24.pp

```
Program Example24;

{ This program demonstrates the TimeStampToDateTime function }

Uses sysutils;
```

```
Var TS : TTimeStamp;
    DT : TDateTime;

Begin
  TS:=DateTimeToTimeStamp (Now);
  With TS do
    begin
    WriteIn ('Now is ',time,' millisecond past midnight');
    WriteIn ('Today is ' ,Date,' days past 1/1/0001');
    end;
  DT:=TimeStampToDateTime(TS);
  WriteIn ('Together this is : ',DateTimeToStr(DT));
End.
```

76.15.276 TimeStampToMSecs

Synopsis: Converts a timestamp to a number of milliseconds.

Declaration:
```
function TimeStampToMSecs(const TimeStamp: TTimeStamp) : Int64
```

Visibility: default

Description: `TimeStampToMSecs` converts `TimeStamp` to the number of milliseconds since `1/1/0001`.

Use `TTimeStamp` variables if you need to keep very precise track of time.

For an example, see MSecsToTimeStamp (1764).

Errors: None.

See also: MSecsToTimeStamp (1764), TimeStampToDateTime (1801)

76.15.277 TimeToStr

Synopsis: Convert a `TDateTime` time to a string using a predefined format.

Declaration:
```
function TimeToStr(Time: TDateTime) : string
function TimeToStr(Time: TDateTime;
                   const FormatSettings: TFormatSettings) : string
```

Visibility: default

Description: `TimeToStr` converts the time in `Time` to a string. It uses the `LongTimeFormat` variable to see what formatting needs to be applied. It is therefor entirely equivalent to a `FormatDateTime('tt',Time)` call.

Note that on unix systems, the localization support must be enabled explicitly, see Localization (1636).

Errors: None.

Listing: ./examples/sysutex/ex25.pp

```
Program Example25;

{ This program demonstrates the TimeToStr function }

Uses sysutils;
```

```
Begin
  WriteIn ('The current time is : ',TimeToStr(Time));
End.
```

76.15.278 Trim

Synopsis: Trim whitespace from the ends of a string.

Declaration:
```
function Trim(const S: string) : string
function Trim(const S: WideString) : WideString
function Trim(const S: UnicodeString) : UnicodeString
```

Visibility: default

Description: `Trim` strips blank characters (spaces and control characters) at the beginning and end of `S` and returns the resulting string. All characters with ordinal values less than or equal to `32` (a space) are stripped.

If the string contains only spaces, an empty string is returned.

Errors: None.

See also: TrimLeft (1803), TrimRight (1804)

Listing: ./examples/sysutex/ex84.pp

```
Program Example84;

{ This program demonstrates the Trim function }

Uses sysutils;
{$H+}

Procedure Testit (S : String);

begin
  WriteIn ('"',Trim(S),'"');
end;

Begin
  Testit ('  ha ha what gets lost ? ');
  Testit (#10#13'haha ');
  Testit ('              ');
End.
```

76.15.279 TrimLeft

Synopsis: Trim whitespace from the beginning of a string.

Declaration:
```
function TrimLeft(const S: string) : string
function TrimLeft(const S: WideString) : WideString
function TrimLeft(const S: UnicodeString) : UnicodeString
```

Visibility: default

Description: `Trim` strips blank characters (spaces and control characters) at the beginning of `S` and returns the resulting string. All characters with ordinal values less than or equal to `32` (a space) are stripped.

If the string contains only spaces, an empty string is returned.

Errors: None.

See also: Trim (1803), TrimRight (1804)

Listing: ./examples/sysutex/ex85.pp

```
Program Example85;

{ This program demonstrates the TrimLeft function }

Uses sysutils;
{$H+}

Procedure Testit (S : String);

begin
  WriteLn ('"',TrimLeft(S),'"');
end;

Begin
  Testit ('  ha ha what gets lost ? ');
  Testit (#10#13'haha ');
  Testit ('              ');
End.
```

76.15.280 TrimRight

Synopsis: Trim whitespace from the end of a string.

Declaration:
```
function TrimRight(const S: string) : string
function TrimRight(const S: WideString) : WideString
function TrimRight(const S: UnicodeString) : UnicodeString
```

Visibility: default

Description: `Trim` strips blank characters (spaces and control characters) at the end of `S` and returns the resulting string. All characters with ordinal values less than or equal to `32` (a space) are stripped.

If the string contains only spaces, an empty string is returned.

Errors: None.

See also: Trim (1803), TrimLeft (1803)

Listing: ./examples/sysutex/ex86.pp

```
Program Example86;

{ This program demonstrates the TrimRight function }

Uses sysutils;
{$H+}

Procedure Testit (S : String);
```

```
begin
  WriteIn ('"',TrimRight(S),'"');
end;

Begin
  Testit ('  ha ha what gets lost ? ');
  Testit (#10#13'haha ');
  Testit ('              ');
End.
```

76.15.281 TryEncodeDate

Synopsis: Try to encode a date, and indicate success.

Declaration:
```
function TryEncodeDate(Year: Word; Month: Word; Day: Word;
                       out Date: TDateTime) : Boolean
```

Visibility: default

Description: `TryEncodeDate` will check the validity of the `Year`, `Month` and `Day` arguments, and if they are all valid, then they will be encoded as a `TDateTime` value and returned in `Date`. The function will return `True` in this case. If an invalid argument is passed, then `False` will be returned.

Errors: None. If an error occurs during the encoding, `False` is returned.

See also: EncodeDate (1704), DecodeDateFully (1700), DecodeDate (1699), TryEncodeTime (1805)

76.15.282 TryEncodeTime

Synopsis: Try to encode a time, and indicate success.

Declaration:
```
function TryEncodeTime(Hour: Word; Min: Word; Sec: Word; MSec: Word;
                       out Time: TDateTime) : Boolean
```

Visibility: default

Description: `TryEncodeTime` will check the validity of the `Hour`, `Min`, `Sec` and `MSec` arguments, and will encode them in a `TDateTime` value which is returned in `Time`. If the arguments are valid, then `True` is returned, otherwise `False` is returned.

Errors: None. If an error occurs during the encoding, `False` is returned.

See also: EncodeTime (1704), DecodeTime (1700), TryEncodeDate (1805)

76.15.283 TryFloatToCurr

Synopsis: Try to convert a float value to a currency value and report on success.

Declaration:
```
function TryFloatToCurr(const Value: Extended; var AResult: Currency)
                        : Boolean
```

Visibility: default

Description: `TryFloatToCurr` tries convert the `Value` floating point value to a `Currency` value. If successful, the function returns `True` and the resulting currency value is returned in `AResult`. It checks whether `Value` is in the valid range of currencies (determined by MinCurrency (1647) and MaxCurrency (1647)). If not, `False` is returned.

Errors: If `Value` is out of range, `False` is returned.

See also: FloatToCurr (1728), MinCurrency (1647), MaxCurrency (1647)

76.15.284 TryStringToGUID

Synopsis: Try to transform a string to a GUID.

Declaration:
```
function TryStringToGUID(const S: string; out Guid: TGuid) : Boolean
```

Visibility: default

Description: `TryStringToGUID` tries to convert the string `S` to a TGUID value, returned in `GUID`. It returns `True` if the conversion succeeds, and `False` if the string `S` does not contain a valid GUID notation. The string `S` must be 38 characters long, must start with { and end on }, and contain a valid GUID string (hex number grouped using 8-4-4-4-12 digits).

Errors: In case S does not contain a valid GUID number, `False` is returned.

See also: StringToGUID (1779)

76.15.285 TryStrToBool

Synopsis: Try to convert a string to a boolean value.

Declaration:
```
function TryStrToBool(const S: string; out Value: Boolean) : Boolean
function TryStrToBool(const S: string; out Value: Boolean;
                      const FormatSettings: TFormatSettings) : Boolean
```

Visibility: default

Description: `TryStrToBool` tries to convert the string `S` to a boolean value, and returns this value in `Value`. In this case, the function returns `True`. If `S` does not contain a valid boolean string, the function returns `False`, and the contents of `Value` is undetermined.

Valid boolean string constants are in the FalseBoolStrs (1664) (for `False` values) and TrueBoolStrs (1668) (for `True` values) variables.

See also: StrToBool (1787), StrToBoolDef (1787)

76.15.286 TryStrToCurr

Synopsis: Try to convert a string to a currency.

Declaration:
```
function TryStrToCurr(const S: string; out Value: Currency) : Boolean
function TryStrToCurr(const S: string; out Value: Currency;
                      const FormatSettings: TFormatSettings) : Boolean
```

Visibility: default

Description: `TryStrToCurr` converts the string `S` to a currency value and returns the value in `Value`. The function returns `True` if it was successful, `False` if not. This is contrary to StrToCurr (1788), which raises an exception when the conversion fails.

The function takes into account locale information.

See also: StrToCurr (1788), TextToFloat (1800)

76.15.287 TryStrToDate

Synopsis: Try to convert a string with a date indication to a TDateTime value.

Declaration:
```
function TryStrToDate(const S: ShortString; out Value: TDateTime)
                      : Boolean
function TryStrToDate(const S: AnsiString; out Value: TDateTime)
                      : Boolean
function TryStrToDate(const S: ShortString; out Value: TDateTime;
                     separator: Char) : Boolean
function TryStrToDate(const S: AnsiString; out Value: TDateTime;
                     separator: Char) : Boolean
function TryStrToDate(const S: ShortString; out Value: TDateTime;
                     const useformat: string; separator: Char) : Boolean
function TryStrToDate(const S: AnsiString; out Value: TDateTime;
                     const useformat: string; separator: Char) : Boolean
function TryStrToDate(const S: string; out Value: TDateTime;
                     const FormatSettings: TFormatSettings) : Boolean
```

Visibility: default

Description: `TryStrToDate` tries to convert the string `S` to a `TDateTime` date value, and stores the date in `Value`. The Date must consist of 1 to three digits, separated by the `DateSeparator` character. If two numbers are given, they are supposed to form the day and month of the current year. If only one number is given, it is supposed to represent the day of the current month. (This is *not* supported in Delphi)

The order of the digits (y/m/d, m/d/y, d/m/y) is determined from the `ShortDateFormat` variable.

The function returns `True` if the string contained a valid date indication, `False` otherwise.

See also: StrToDate (1788), StrToTime (1795), TryStrToTime (1809), TryStrToDateTime (1807), DateToStr (1698), TimeToStr (1802)

76.15.288 TryStrToDateTime

Synopsis: Try to convert a string with date/time indication to a TDateTime value.

Declaration:
```
function TryStrToDateTime(const S: ShortString; out Value: TDateTime)
                          : Boolean
function TryStrToDateTime(const S: AnsiString; out Value: TDateTime)
                          : Boolean
function TryStrToDateTime(const S: string; out Value: TDateTime;
                         const FormatSettings: TFormatSettings)
                          : Boolean
```

Visibility: default

Description: `TryStrToDateTime` tries to convert the string `S` to a `TDateTime` date and time value, and stores the result in `Value`. The date must consist of 1 to three digits, separated by the `DateSeparator` character. If two numbers are given, they are supposed to form the day and month of the current year. If only one number is given, it is supposed to represent the day of the current month (This is *not* supported in Delphi). The time must consist of 1 to 4 digits, separated by the `TimeSeparator` character. If two numbers are given, they are supposed to form the hour and minutes.

The function returns `True` if the string contained a valid date and time indication, `False` otherwise.

See also: TryStrToDate (1807), TryStrToTime (1809), StrToDateTime (1790), StrToTime (1795), DateToStr (1698), TimeToStr (1802)

76.15.289 TryStrToDWord

Synopsis: Try to convert a string to DWord (cardinal).

Declaration:
```
function TryStrToDWord(const s: string; out D: DWord) : Boolean
```

Visibility: default

Description: `TryStrToDWord` will try to convert the string `S` to a `DWord` value. It returns `True` if the conversion was successful, and in that case returns the value in `D`.

If the conversion failed or the value was outside of the valid range for DWord values, `False` is returned.

See also: StrToDWord (1791), StrToDWordDef (1791)

76.15.290 TryStrToFloat

Synopsis: Try to convert a string to a float.

Declaration:
```
function TryStrToFloat(const S: string; out Value: Single) : Boolean
function TryStrToFloat(const S: string; out Value: Single;
                       const FormatSettings: TFormatSettings) : Boolean
function TryStrToFloat(const S: string; out Value: Double) : Boolean
function TryStrToFloat(const S: string; out Value: Double;
                       const FormatSettings: TFormatSettings) : Boolean
function TryStrToFloat(const S: string; out Value: Extended) : Boolean
function TryStrToFloat(const S: string; out Value: Extended;
                       const FormatSettings: TFormatSettings) : Boolean
```

Visibility: default

Description: `TryStrToFloat` tries to convert the string `S` to a floating point value, and stores the result in `Value`. It returns `True` if the operation was successful, and `False` if it failed. This operation takes into account the system settings for floating point representations.

Errors: On error, `False` is returned.

See also: StrToFloat (1791)

76.15.291 TryStrToInt

Synopsis: Try to convert a string to an integer, and report on success.

Declaration:
```
function TryStrToInt(const s: string; out i: LongInt) : Boolean
```

Visibility: default

Description: `TryStrToInt` tries to convert the string `S` to an integer, and returns `True` if this was successful. In that case the converted integer is returned in `I`. If the conversion failed, (an invalid string, or the value is out of range) then `False` is returned.

Errors: None. On error, `False` is returned.

See also: StrToInt (1793), TryStrToUint (1810), TryStrToInt64 (1809), StrToIntDef (1794), StrToInt64 (1794), StrToInt64Def (1794)

76.15.292 TryStrToInt64

Synopsis: Try to convert a string to an int64 value, and report on success.

Declaration:
```
function TryStrToInt64(const s: string; out i: Int64) : Boolean
```

Visibility: default

Description: `TryStrToInt64` tries to convert the string `S` to a Int64 value, and returns this value in `I` if successful. If the conversion was successful, the function result is `True`, or `False` otherwise. The string can only contain numerical characters, and optionally a minus sign as the first character. Whitespace is not allowed.

Hexadecimal values (starting with the $ character) are supported.

Errors: None. On error, `False` is returned.

See also: StrToInt64 (1794), StrToInt64Def (1794), StrToInt (1793), TryStrToInt (1808), StrToIntDef (1794)

76.15.293 TryStrToQWord

Synopsis: Try to convert a string to a QWord value, and report on success.

Declaration:
```
function TryStrToQWord(const s: string; out Q: QWord) : Boolean
```

Visibility: default

Description: `TryStrToQWord` tries to convert the string `S` to a valid QWord (unsigned 64-bit) value, and stores the result in `I`. If the conversion fails, the function returns `False`, else it returns `True`.

See also: StrToQWord (1795), StrToQWordDef (1795), TryStrToInt64 (1809), TryStrToInt (1808)

76.15.294 TryStrToTime

Synopsis: Try to convert a string with a time indication to a TDateTime value.

Declaration:
```
function TryStrToTime(const S: ShortString; out Value: TDateTime)
                       : Boolean
function TryStrToTime(const S: AnsiString; out Value: TDateTime)
                       : Boolean
function TryStrToTime(const S: ShortString; out Value: TDateTime;
                      separator: Char) : Boolean
function TryStrToTime(const S: AnsiString; out Value: TDateTime;
                      separator: Char) : Boolean
function TryStrToTime(const S: string; out Value: TDateTime;
                      const FormatSettings: TFormatSettings) : Boolean
```

Visibility: default

Description: `TryStrToTime` tries to convert the string `S` to a `TDateTime` time value, and stores the result in `Value`. The time must consist of 1 to 4 digits, separated by the `TimeSeparator` character. If two numbers are given, they are supposed to form the hour and minutes.

The function returns `True` if the string contained a valid time indication, `False` otherwise.

See also: TryStrToDate (1807), TryStrToDateTime (1807), StrToDate (1788), StrToTime (1795), DateToStr (1698), TimeToStr (1802)

76.15.295 TryStrToUInt

Synopsis: Try to convert a string to an unsigned integer, and report on success.

Declaration:
```
function TryStrToUInt(const s: string; out C: Cardinal) : Boolean
```

Visibility: default

Description: `TryStrToUInt` tries to convert the string `S` to an unsigned 32-bit integer (a `Cardinal`), and returns `True` if this was successful. In that case the converted unsigned integer is returned in `C`. If the conversion failed, (an invalid string, or the value is out of range) then `False` is returned.

Errors: None. On error, `False` is returned.

See also: StrToInt (1793), TryStrToInt (1808), TryStrToInt64 (1809), StrToIntDef (1794), StrToInt64 (1794), StrToInt64Def (1794)

76.15.296 TryStrToUInt64

Synopsis: Try to convert a string to an unsigned 64-bit integer, and report on success.

Declaration:
```
function TryStrToUInt64(const s: string; out u: UInt64) : Boolean
```

Visibility: default

Description: `TryStrToUInt` tries to convert the string `S` to an unsigned 64-bit integer (a `Cardinal`), and returns `True` if this was successful. In that case the converted unsigned integer is returned in `U`. If the conversion failed, (an invalid string, or the value is out of range) then `False` is returned.

Errors: None. On error, `False` is returned.

See also: StrToInt (1793), TryStrToInt (1808), TryStrToInt64 (1809), StrToIntDef (1794), StrToInt64 (1794), StrToInt64Def (1794)

76.15.297 UIntToStr

Synopsis: Unsigned integer to String.

Declaration:
```
function UIntToStr(Value: QWord) : string
function UIntToStr(Value: Cardinal) : string
```

Visibility: default

Description: `UIntToStr` is the unsigned counterpart to IntToStr (1758): it converts the unsigned integer `Value` to its string representation. The resulting string has only as much characters as needed to represent the value.

Errors: None

See also: IntToStr (1758)

76.15.298 UnhookSignal

Synopsis: UnHook a specified signal.

Declaration:
```
procedure UnhookSignal(RtlSigNum: Integer; OnlyIfHooked: Boolean)
```

Visibility: default

Description: `UnHookSignal` de-installs the RTL default signal handler for signal `RtlSigNum`. If `OnlyIfHooked` is `True` then `UnhookSignal` will first check if the signal was hooked by the RTL routines, and has not been overridden since.

76.15.299 UnicodeCompareStr

Synopsis: Compare 2 Unicode strings.

Declaration:
```
function UnicodeCompareStr(const s1: UnicodeString;
                           const s2: UnicodeString) : PtrInt
```

Visibility: default

Description: `UnicodeCompareStr` compares 2 Unicode strings `S1` and `S2` in a case sensitive manner. The result of the function is

< 0 If `S1<S2`

0If `S1=S2`

> 0 If `S1>S2`

This function relies on a widestring manager to perform the actual comparison, as it will take into account various equivalent code points: it is not a simple byte-by-byte comparison.

See also: UnicodeCompareText (1811), CompareStr (1689), CompareText (1690), UnicodeSameStr (1813)

76.15.300 UnicodeCompareText

Synopsis: Compare 2 strings case insensitively.

Declaration:
```
function UnicodeCompareText(const s1: UnicodeString;
                            const s2: UnicodeString) : PtrInt
```

Visibility: default

Description: `UnicodeCompareText` compares 2 Unicode strings `S1` and `S2`, ignoring case. The result of the function is

< 0 If `S1<S2`

0If `S1=S2`

> 0 If `S1>S2`

This function relies on a widestring manager to perform the actual comparison, as it will take into account various equivalent code points: it is not a simple byte-by-byte comparison.

See also: UnicodeCompareText (1811), CompareStr (1689), CompareText (1690), UnicodeSameStr (1813)

76.15.301 UnicodeFmtStr

Synopsis: Format a string with given arguments, procedural version.

Declaration:
```
procedure UnicodeFmtStr(var Res: UnicodeString;
                        const Fmt: UnicodeString;
                        const args: Array of const)
procedure UnicodeFmtStr(var Res: UnicodeString;
                        const Fmt: UnicodeString;
                        const args: Array of const;
                        const FormatSettings: TFormatSettings)
```

Visibility: default

Description: `UnicodeFmtStr` calls UnicodeFormat (1812) with `Fmt` and `Args` as arguments, and stores the result in `Res`. For more information on how the resulting string is composed, see UnicodeFormat (1812).

Errors: In case of error, an EConvertError (1830) exception is raised.

See also: UnicodeFormat (1812), UnicodeFormatBuf (1812)

76.15.302 UnicodeFormat

Synopsis: Format Unicode string.

Declaration:
```
function UnicodeFormat(const Fmt: UnicodeString;
                       const Args: Array of const) : UnicodeString
function UnicodeFormat(const Fmt: UnicodeString;
                       const Args: Array of const;
                       const FormatSettings: TFormatSettings)
                        : UnicodeString
```

Visibility: default

Description: `UnicodeFormat` is the Unicode equivalent of Format (1735). It follows the same rules and uses the same formatting strings.

Errors: In case of error, an EConvertError (1830) exception is raised.

See also: Format (1735), UnicodeFormatBuf (1812)

76.15.303 UnicodeFormatBuf

Synopsis: Format a Unicode string with given arguments and store the result in a unicodebuffer.

Declaration:
```
function UnicodeFormatBuf(var Buffer; BufLen: Cardinal; const Fmt;
                          fmtLen: Cardinal; const Args: Array of const)
                           : Cardinal
function UnicodeFormatBuf(var Buffer; BufLen: Cardinal; const Fmt;
                          fmtLen: Cardinal; const Args: Array of const;
                          const FormatSettings: TFormatSettings)
                           : Cardinal
```

Visibility: default

Description: `UnicodeFormatBuf` is the Unicode equivalent of FormatBuf (1742). It follows the same rules and uses the same formatting st

Errors: In case of error, an EConvertError (1830) exception is raised.

See also: FormatBuf (1742), UnicodeFormat (1812)

76.15.304 UnicodeLowerCase

Synopsis: Return lowercase version of a string.

Declaration:
```
function UnicodeLowerCase(const s: UnicodeString) : UnicodeString
```

Visibility: default

Description: `UnicodeLowerCase` returns an all-lowercase version of the unicodestring `S`. It relies on the Unicode manager to do so.

See also: UnicodeUpperCase (1814), UpperCase (1814), LowerCase (1763)

76.15.305 UnicodeSameStr

Synopsis: Check whether 2 strings are equal.

Declaration:
```
function UnicodeSameStr(const s1: UnicodeString;
                        const s2: UnicodeString) : Boolean
```

Visibility: default

Description: `UnicodeSameStr` checks whether the Unicode strings `S1` and `S2` are equal, case sensitively. The function returns `True` if the strings are equal, `False` if they are not. This function relies on a widestring manager to perform the actual comparison, as it will take into account various equivalent code points: it is not a simple byte-by-byte comparison.

See also: UnicodeCompareText (1811), CompareStr (1689), CompareText (1690), UnicodeSameText (1813), SameStr (1769)

76.15.306 UnicodeSameText

Synopsis: Check whether 2 strings are the same, ignoring case.

Declaration:
```
function UnicodeSameText(const s1: UnicodeString;
                         const s2: UnicodeString) : Boolean
```

Visibility: default

Description: `UnicodeSameText` checks whether the Unicode strings `S1` and `S2` are equal, ignoring case. The function returns `True` if the strings are equal, `False` if they are not. This function relies on a widestring manager to perform the actual comparison, as it will take into account various equivalent code points: it is not a simple byte-by-byte comparison.

See also: UnicodeCompareText (1811), CompareStr (1689), CompareText (1690), UnicodeSameStr (1813), SameText (1769)

76.15.307 UnicodeStringReplace

Synopsis: Replace one occurrence of a string with another.

Declaration:
```
function UnicodeStringReplace(const S: UnicodeString;
                              const OldPattern: UnicodeString;
                              const NewPattern: UnicodeString;
                              Flags: TReplaceFlags) : UnicodeString
function UnicodeStringReplace(const S: UnicodeString;
                              const OldPattern: UnicodeString;
                              const NewPattern: UnicodeString;
                              Flags: TReplaceFlags; out aCount: Integer)
                               : UnicodeString
```

Visibility: default

Description: `UnicodeStringReplace` is the Unicode version of StringReplace (1778); it follows the same rules and has the same behaviour, but operates on Unicode strings instead of ansistrings.

See also: StringReplace (1778)

76.15.308 UnicodeUpperCase

Synopsis: Return uppercase version of a string.

Declaration:
```
function UnicodeUpperCase(const s: UnicodeString) : UnicodeString
```

Visibility: default

Description: `UnicodeUpperCase` returns an all-uppercase version of the unicodestring `S`. It relies on the Unicode manager to do so.

See also: UnicodeLowerCase (1812), LowerCase (1763), UpperCase (1814)

76.15.309 UniversalTimeToLocal

Declaration:
```
function UniversalTimeToLocal(UT: TDateTime) : TDateTime
function UniversalTimeToLocal(UT: TDateTime; TZOffset: Integer)
                                : TDateTime
```

Visibility: default

76.15.310 UniversalToFileDate

Declaration:
```
function UniversalToFileDate(DateTime: TDateTime) : Int64
```

Visibility: default

76.15.311 UpperCase

Synopsis: Return an uppercase version of a string.

Declaration:
```
function UpperCase(const s: string) : string;  Overload
function UpperCase(const s: string; LocaleOptions: TLocaleOptions)
                   : string;  Overload
function UpperCase(const s: UnicodeString) : UnicodeString;  Overload
```

Visibility: default

Description: `UpperCase` returns the uppercase equivalent of `S`. Ansi characters are not taken into account, only ASCII codes below 127 are converted. It is completely equivalent to the `UpCase` function of the system unit, and is provided for compatibility only.

Errors: None.

See also: AnsiLowerCase (1674), LowerCase (1763), AnsiUpperCase (1682)

Listing: ./examples/sysutex/ex87.pp

```
Program Example87;

{ This program demonstrates the UpperCase function }

Uses sysutils;

Begin
  WriteLn (UpperCase('this  will come OUT ALL uPpErCaSe !'));
End.
```

76.15.312 VendorName

Synopsis: Return Application vendor Name.

Declaration: `function VendorName : string`

Visibility: default

Description: `VendorName` returns the application vendor name. In order to set the application vendor name, the OnGetVendorName (1666) event must be set, and an appropriate return value must be returned. The Vendor name is used in GetAppConfigDir (1746) and GetAppConfigFile (1746) to determine the configuration directory.

Errors: If OnGetVendorName (1666) is not set, an empty string is returned.

See also: OnGetVendorName (1666), GetAppConfigDir (1746), GetAppConfigFile (1746)

76.15.313 WideBytesOf

Synopsis: Returns the contents of a widestring as an array of bytes.

Declaration: `function WideBytesOf(const Value: UnicodeString) : TBytes`

Visibility: default

Description: `WideBytesOf` returns the contents of the widestring `Value` as an array of bytes. The array will have as length twice the length of the wide string, as each wide character contains 2 bytes.

See also: StringOf (1778)

76.15.314 WideCompareStr

Synopsis: Compare two widestrings (case sensitive).

Declaration:
```
function WideCompareStr(const s1: WideString; const s2: WideString)
                        : PtrInt
```

Visibility: default

Description: `WideCompareStr` compares two widestrings and returns the following result:

< 0if `S1<S2`.

0if `S1=S2`.

> 0if `S1>S2`.

The comparison takes into account wide characters, i.e. it takes care of strange accented characters. Contrary to WideCompareText (1816), the comparison is case sensitive.

Errors: None.

See also: WideCompareText (1816), WideSameStr (1817), WideSameText (1817)

76.15.315 WideCompareText

Synopsis: Compare two widestrings (ignoring case).

Declaration:
```
function WideCompareText(const s1: WideString; const s2: WideString)
                         : PtrInt
```

Visibility: default

Description: `WideCompareStr` compares two widestrings and returns the following result:

< 0if `S1<S2`.

0if `S1=S2`.

> 0if `S1>S2`.

The comparison takes into account wide characters, i.e. it takes care of strange accented characters. Contrary to WideCompareStr (1815), the comparison is case insensitive.

Errors: None.

See also: WideCompareStr (1815), WideSameStr (1817), WideSameText (1817)

76.15.316 WideFmtStr

Synopsis: Widestring format.

Declaration:
```
procedure WideFmtStr(var Res: WideString; const Fmt: WideString;
                     const args: Array of const)
procedure WideFmtStr(var Res: WideString; const Fmt: WideString;
                     const args: Array of const;
                     const FormatSettings: TFormatSettings)
```

Visibility: default

Description: `WideFmtStr` formats `Args` according to the format string in `Fmt` and returns the resulting string in `Res`.

See also: WideFormat (1816), WideFormatBuf (1817), Format (1735)

76.15.317 WideFormat

Synopsis: Format a wide string.

Declaration:
```
function WideFormat(const Fmt: WideString; const Args: Array of const)
                    : WideString
function WideFormat(const Fmt: WideString; const Args: Array of const;
                    const FormatSettings: TFormatSettings) : WideString
```

Visibility: default

Description: `WideFormat` does the same as Format (1735) but accepts as a formatting string a WideString. The resulting string is also a WideString.

For more information about the used formatting characters, see the Format (1735) string.

See also: Format (1735)

76.15.318 WideFormatBuf

Synopsis: Format widestring in a buffer.

Declaration:
```
function WideFormatBuf(var Buffer; BufLen: Cardinal; const Fmt;
                       fmtLen: Cardinal; const Args: Array of const)
                        : Cardinal
function WideFormatBuf(var Buffer; BufLen: Cardinal; const Fmt;
                       fmtLen: Cardinal; const Args: Array of const;
                       const FormatSettings: TFormatSettings) : Cardinal
```

Visibility: default

Description: `WideFormatBuf` calls simply WideFormat (1816) with `Fmt` (with length `FmtLen` bytes) and stores maximum `BufLen` bytes in the buffer `buf`. It returns the number of copied bytes.

See also: WideFmtStr (1816), WideFormat (1816), Format (1735), FormatBuf (1742)

76.15.319 WideLowerCase

Synopsis: Change a widestring to all-lowercase.

Declaration:
```
function WideLowerCase(const s: WideString) : WideString
```

Visibility: default

Description: `WideLowerCase` converts the string `S` to lowercase characters and returns the resulting string. It takes into account the operating system language settings when doing this, so special characters are converted correctly as well.

Remark On Unix-like platforms, a widestring manager must be installed for this function to work correctly.

Errors: None.

See also: WideUpperCase (1819)

76.15.320 WideSameStr

Synopsis: Check whether two widestrings are the same (case sensitive).

Declaration:
```
function WideSameStr(const s1: WideString; const s2: WideString)
                      : Boolean
```

Visibility: default

Description: `WideSameStr` returns `True` if WideCompareStr (1815) returns 0 (zero), i.e. when `S1` and `S2` are the same string (taking into account case).

See also: WideSameText (1817), WideCompareStr (1815), WideCompareText (1816), AnsiSameStr (1675)

76.15.321 WideSameText

Synopsis: Check whether two widestrings are the same (ignoring case).

Declaration:
```
function WideSameText(const s1: WideString; const s2: WideString)
                      : Boolean
```

Visibility: default

Description: `WideSameText` returns `True` if WideCompareText (1816) returns 0 (zero), i.e. when `S1` and `S2` are the same string (taking into account case).

See also: WideSameStr (1817), WideCompareStr (1815), WideCompareText (1816), AnsiSameText (1675)

76.15.322 WideStrAlloc

Synopsis: Allocate a null-terminated widestring on the heap.

Declaration:
```
function WideStrAlloc(size: Cardinal) : PWideChar
```

Visibility: default

Description: `WideStrAlloc` reserves memory on the heap for a widestring with length `Len`, (terminating `#0#0` included), and returns a pointer to it.

Additionally, `WideStrAlloc` allocates 4 extra bytes to store the size of the allocated memory.

Errors: None.

See also: StrBufSize (1772), StrDispose (1775), StrAlloc (1310)

76.15.323 WideStringOf

Synopsis: Create Unicode string from array of bytes.

Declaration:
```
function WideStringOf(const Value: TBytes) : UnicodeString
```

Visibility: default

Description: `WideStringOf` converts an array of bytes (`Bytes`) to a Unicode string. It considers each pair of bytes in the array as a single wide char. The array should have an even length. If the length is uneven, the last byte will be ignored.

This function performs the opposite operation of BytesOf (1686).

To create a string where the bytes are interpreted as ansichars, use StringOf (1778) instead.

See also: StringOf (1778), WideBytesOf (1815), BytesOf (1686)

76.15.324 WideStringReplace

Synopsis: Replace occurrences of one substring with another in a widestring.

Declaration:
```
function WideStringReplace(const S: WideString;
                           const OldPattern: WideString;
                           const NewPattern: WideString;
                           Flags: TReplaceFlags) : WideString
function WideStringReplace(const S: WideString;
                           const OldPattern: WideString;
                           const NewPattern: WideString;
                           Flags: TReplaceFlags; out aCount: Integer)
                            : WideString
```

Visibility: default

Description: `WideStringReplace` searches the string `S` for occurrences of the string `OldPattern` and, if it is found, replaces it with `NewPattern`. It returns the resulting string. The behaviour of `StringReplace` can be tuned with `Flags`, which is of type TReplaceFlags (1659). Standard behaviour is to replace only the first occurrence of `OldPattern`, and to search case sensitively.

Errors: None.

See also: TReplaceFlags (1659), StringReplace (1778)

76.15.325 WideUpperCase

Synopsis: Change a widestring to all-uppercase.

Declaration:
```
function WideUpperCase(const s: WideString) : WideString
```

Visibility: default

Description: `WideUpperCase` converts the string `S` to uppercase characters and returns the resulting string. It takes into account the operating system language settings when doing this, so special characters are converted correctly as well.

Remark On Unix-like platforms, a widestring manager must be installed for this function to work correctly.

Errors: None.

See also: WideLowerCase (1817)

76.15.326 WrapText

Synopsis: Word-wrap a text.

Declaration:
```
function WrapText(const Line: string; const BreakStr: string;
                  const BreakChars: TSysCharSet; MaxCol: Integer)
                   : string
function WrapText(const Line: string; MaxCol: Integer) : string
```

Visibility: default

Description: `WrapText` does a wordwrap at column `MaxCol` of the string in `Line`. It breaks the string only at characters which are in `BreakChars` (default whitespace and hyphen) and inserts then the string `BreakStr` (default the lineending character for the current OS).

Portions of text that are between single or double quotes will be considered single words and the text will not be broken between the quotes.

See also: StringReplace (1778)

76.16 TDateTimeInfoRec

```
TDateTimeInfoRec = record
private
  data : tstatxplatform;
  function
  GetCreationTime : TDateTime;
  function GetLastAccessTime : TDateTime
  ;
  function GetTimeStamp : TDateTime;
public
  property CreationTime
  : TDateTime;
  property LastAccessTime : TDateTime;
```

```
  property TimeStamp
  : TDateTime;
end
```

76.16.1 Property overview

Page	Properties	Access	Description
1820	CreationTime	r	
1820	LastAccessTime	r	
1820	TimeStamp	r	

76.16.2 TDateTimeInfoRec.CreationTime

Declaration: `Property CreationTime : TDateTime`

Visibility: public

Access: Read

76.16.3 TDateTimeInfoRec.LastAccessTime

Declaration: `Property LastAccessTime : TDateTime`

Visibility: public

Access: Read

76.16.4 TDateTimeInfoRec.TimeStamp

Declaration: `Property TimeStamp : TDateTime`

Visibility: public

Access: Read

76.17 TOSVersion

```
TOSVersion = record
public
  TArchitecture = (arIntelX86,arIntelX64
  ,arARM32,arARM64,arIntelX16,
                   arXTensa,arAVR,arM68k
  ,arPowerPC,arPower64,arMips,
                   arMipsel,arMips64
  ,arMips64el,arJVM,arWasm32,arSparc,
                   arSparc64,arRiscV32
  ,arRiscV64,arZ80,arLoongArch64,
                   arOther);
  TPlatform
  = (pfWindows,pfMacOS,pfiOS,pfAndroid,pfWinRT,pfLinux,pfGo32v2,
  pfOS2,pfFreeBSD,pfBeos,pfNetBSD,pfAmiga,pfAtari,pfSolaris
```

```
  ,
               pfQNX,pfNetware,pfOpenBSD,pfWDosX,pfPalmos,pfMacOSClassic
  ,
               pfDarwin,pfEMX,pfWatcom,pfMorphos,pfNetwLibC,pfWinCE
  ,pfGBA,
               pfNDS,pfEmbedded,pfSymbian,pfHaiku,pfIPhoneSim
  ,pfAIX,
               pfJava,pfNativeNT,pfMSDos,pfWII,pfAROS,pfDragonFly
  ,pfWin16,
               pfFreeRTOS,pfZXSpectrum,pfMSXDOS,pfAmstradCPC
  ,pfSinclairQL,
               pfWasi,pfOther);
  AllArchitectures
  = [Low(TArchitecture)..High(TArchitecture)];
  AllPlatforms = [Low
  (TPlatform)..High(TPlatform)];
private
  FFull : string;
  FArchitecture
  : TArchitecture;
  FBuild : Integer;
  FMajor : Integer;
  FMinor
  : Integer;
  FName : string;
  FPlatform : TPlatform;
  FServicePackMajor
  : Integer;
  FServicePackMinor : Integer;
  FPrettyName : string
  ;
  FLibCVersionMajor : Integer;
  FLibCVersionMinor : Integer;
  class constructor Create;
public
  Check;
  class function ToString
  : string;  Static;
  property Architecture : TArchitecture;
  property
  Build : Integer;
  property Major : Integer;
  property Minor : Integer
  ;
  property Name : string;
  property Platform : TPlatform;
  property
  ServicePackMajor : Integer;
  property ServicePackMinor : Integer
  ;
  property PrettyName : string;
  property LibCVersionMajor : Integer
  ;
  property LibCVersionMinor : Integer;
```

```
end
```

76.17.1 Method overview

Page	Method	Description
1822	Check	
1822	ToString	

76.17.2 Property overview

Page	Properties	Access	Description
1822	Architecture	r	
1822	Build	r	
1824	LibCVersionMajor	r	
1824	LibCVersionMinor	r	
1823	Major	r	
1823	Minor	r	
1823	Name	r	
1823	Platform	r	
1824	PrettyName	r	
1823	ServicePackMajor	r	
1823	ServicePackMinor	r	

76.17.3 TOSVersion.Check

Declaration:
```
class function Check(AMajor: Integer) : Boolean;  Overload;  Static
class function Check(AMajor: Integer; AMinor: Integer) : Boolean
                    ;  Overload;  Static
class function Check(AMajor: Integer; AMinor: Integer;
                    AServicePackMajor: Integer) : Boolean;  Overload
                    ;  Static
```

Visibility: public

76.17.4 TOSVersion.ToString

Declaration:
```
class function ToString : string;  Static
```

Visibility: public

76.17.5 TOSVersion.Architecture

Declaration:
```
Property Architecture : TArchitecture
```

Visibility: public

Access: Read

76.17.6 TOSVersion.Build

Declaration:
```
Property Build : Integer
```

Visibility: public

Access: Read

76.17.7 TOSVersion.Major

Declaration: `Property Major : Integer`

Visibility: public

Access: Read

76.17.8 TOSVersion.Minor

Declaration: `Property Minor : Integer`

Visibility: public

Access: Read

76.17.9 TOSVersion.Name

Declaration: `Property Name : string`

Visibility: public

Access: Read

76.17.10 TOSVersion.Platform

Declaration: `Property Platform : TPlatform`

Visibility: public

Access: Read

76.17.11 TOSVersion.ServicePackMajor

Declaration: `Property ServicePackMajor : Integer`

Visibility: public

Access: Read

76.17.12 TOSVersion.ServicePackMinor

Declaration: `Property ServicePackMinor : Integer`

Visibility: public

Access: Read

76.17.13 TOSVersion.PrettyName

Declaration: `Property PrettyName : string`

Visibility: public

Access: Read

76.17.14 TOSVersion.LibCVersionMajor

Declaration: `Property LibCVersionMajor : Integer`

Visibility: public

Access: Read

76.17.15 TOSVersion.LibCVersionMinor

Declaration: `Property LibCVersionMinor : Integer`

Visibility: public

Access: Read

76.18 TRawbyteSearchRec

```
TRawbyteSearchRec = record
public
  Time : Int64deprecated;
  Size
  : Int64;
  Attr : LongInt;
  Name : RawByteString;
  ExcludeAttr
  : LongInt;
  FindHandle : Pointer;
  Mode : TMode;
private
  function
  GetTimeStamp : TDateTime;
  function GetTimeStampUTC : TDateTime
  ;
public
  function IsDirectory : Boolean;
  function IsCurrentOrParentDir
  : Boolean;
  property TimeStamp : TDateTime;
  property TimeStampUTC
  : TDateTime;
end
```

`TRawbyteSearchRec` is a search handle description record using single-byte strings. It is initialized by a call to FindFirst (1727) and can be used to do subsequent calls to FindNext (1728). It

contains the result of these function calls. It must be used to close the search sequence with a call to FindClose (1726).

Remark Not all fields of this record should be used. Some of the fields are for internal use only. (`PathOnly` for example, is only provided for Kylix compatibility)

Remark Note that for files with Unicode filenames this is a converted value from the Unicode filename. Depending on the codepage, this may or may not be a correct rendering of the correct Unicode filename.

76.18.1 Method overview

Page	Method	Description
1825	IsCurrentOrParentDir	
1825	IsDirectory	

76.18.2 Property overview

Page	Properties	Access	Description
1825	TimeStamp	r	
1825	TimeStampUTC	r	

76.18.3 TRawbyteSearchRec.IsDirectory

Declaration: `function IsDirectory : Boolean`

Visibility: public

76.18.4 TRawbyteSearchRec.IsCurrentOrParentDir

Declaration: `function IsCurrentOrParentDir : Boolean`

Visibility: public

76.18.5 TRawbyteSearchRec.TimeStamp

Synopsis:

Declaration: `Property TimeStamp : TDateTime`

Visibility: public

Access: Read

Description: `TimeStamp` is the file timestamp (`Date`) in `TDateTime` format, as opposed to file system timestamp.

See also: TRawbyteSearchRec.Date (1824)

76.18.6 TRawbyteSearchRec.TimeStampUTC

Declaration: `Property TimeStampUTC : TDateTime`

Visibility: public

Access: Read

76.19 TRawbyteSymLinkRec

```
TRawbyteSymLinkRec = record
public
  TargetName : RawByteString;
  Size : Int64;
  Attr : LongInt;
  Mode : TMode;
private
  function
  GetTimeStamp : TDateTime;
public
  property TimeStamp : TDateTime
  ;
end
```

`TSymLinkRec` contains information about the target of a symlink. It is used in FileGetSymLinkTarget (1720) to return information. The following fields are present:

TargetName Target file filename in single-byte string.

Size Target file size.

Attr Target file attributes.

Mode #ShortDescr:TRawbyteSymLinkRec.Mode

TimeStamp Target file timestamp.

76.19.1 Property overview

Page	Properties	Access	Description
1826	TimeStamp	r	Target file timestamp.

76.19.2 TRawbyteSymLinkRec.TimeStamp

Synopsis: Target file timestamp.

Declaration: `Property TimeStamp : TDateTime`

Visibility: public

Access: Read

76.20 TUnicodeSearchRec

```
TUnicodeSearchRec = record
public
  Time : Int64deprecated;
  Size
  : Int64;
  Attr : LongInt;
  Name : UnicodeString;
```

```
  ExcludeAttr
  : LongInt;
  FindHandle : Pointer;
  Mode : TMode;
private
  function
  GetTimeStamp : TDateTime;
  function GetTimeStampUTC : TDateTime
  ;
public
  function IsDirectory : Boolean;
  function IsCurrentOrParentDir
  : Boolean;
  property TimeStamp : TDateTime;
  property TimeStampUTC
  : TDateTime;
end
```

TRawbyteSearchRec is a search handle description record using multi-byte strings. It is initialized by a call to FindFirst (1727) and can be used to do subsequent calls to FindNext (1728). It contains the result of these function calls. It must be used to close the search sequence with a call to FindClose (1726).

Remark Not all fields of this record should be used. Some of the fields are for internal use only. (PathOnly for example, is only provided for Kylix compatibility)

76.20.1 Method overview

Page	Method	Description
1827	IsCurrentOrParentDir	
1827	IsDirectory	

76.20.2 Property overview

Page	Properties	Access	Description
1827	TimeStamp	r	Time stamp as datetime.
1828	TimeStampUTC	r	

76.20.3 TUnicodeSearchRec.IsDirectory

Declaration: `function IsDirectory : Boolean`

Visibility: public

76.20.4 TUnicodeSearchRec.IsCurrentOrParentDir

Declaration: `function IsCurrentOrParentDir : Boolean`

Visibility: public

76.20.5 TUnicodeSearchRec.TimeStamp

Synopsis: Time stamp as datetime.

Declaration: `Property TimeStamp : TDateTime`

Visibility: public

Access: Read

Description: `TimeStamp` is the file timestamp (`Date`) in `TDateTime` format, as opposed to file system timestamp.

See also: TRawbyteSearchRec.Date (1824)

76.20.6 TUnicodeSearchRec.TimeStampUTC

Declaration: `Property TimeStampUTC : TDateTime`

Visibility: public

Access: Read

76.21 TUnicodeSymLinkRec

```
TUnicodeSymLinkRec = record
public
  TargetName : UnicodeString;
  Attr : LongInt;
  Size : Int64;
  Mode : TMode;
private
  function
  GetTimeStamp : TDateTime;
public
  property TimeStamp : TDateTime
  ;
end
```

`TUnicodeSymLinkRec` contains information about the target of a symlink. It is used in FileGetSymLinkTarget (1720) to return information. The following fields are present:

TargetName Target file filename in single-byte string.

Size Target file size.

Attr Target file attributes.

Mode **#ShortDescr:TRawbyteSymLinkRec.Mode**

TimeStamp Target file timestamp.

76.21.1 Property overview

Page	Properties	Access	Description
1829	TimeStamp	r	Target file timestamp.

76.21.2 TUnicodeSymLinkRec.TimeStamp

Synopsis: Target file timestamp.

Declaration: `Property TimeStamp : TDateTime`

Visibility: public

Access: Read

76.22 EAbort

76.22.1 Description

`Abort` is raised by the Abort (1668) procedure. It is not displayed in GUI applications, and serves only to immediately abort the current procedure, and return control to the main program loop.

See also: Abort (1668)

76.23 EAbstractError

76.23.1 Description

`EAbstractError` is raised when an abstract error occurs, i.e. when an unimplemented abstract method is called.

76.24 EAccessViolation

76.24.1 Description

`EAccessViolation` is raised when the OS reports an Access Violation, i.e. when invalid memory is accessed.

See also: EObjectCheck (1836)

76.25 EArgumentException

76.25.1 Description

`EArgumentException` is raised by many character conversion/handling routines to indicate an erroneous argument was passed to the function (usually indicating an invalid codepoint in a Unicode string).

See also: EArgumentOutOfRangeException (1830)

76.26 EArgumentNilException

76.26.1 Description

`EArgumentNilException` is an exception raised when an argument is `Nil` when it should not be `Nil`.

This exception class is provided for Delphi compatibility, but is not actually used in FPC.

See also: Exception (1839)

76.27 EArgumentOutOfRangeException

76.27.1 Description

`EArgumentOutOfRangeException` is raised by many character conversion/handling routines to indicate an erroneous argument was passed to the function (indicating an invalid character index in a Unicode string).

See also: EArgumentException (1829)

76.28 EAssertionFailed

76.28.1 Description

`EAssertionFailed` is raised when an application that is compiled with assertions, encounters an invalid assertion.

76.29 EBusError

76.29.1 Description

`EBusError` is raised in case of a bus error.

76.30 ECFError

76.31 EControlC

76.31.1 Description

`EControlC` is raised when the user has pressed CTRL-C in a console application.

76.32 EConvertError

76.32.1 Description

`EConvertError` is raised by the various conversion routines in the SysUtils unit. The message will contain more specific error information.

76.33 EDirectoryNotFoundException

76.33.1 Description

`EDirectoryNotFoundException` is an exception raised when a directory is referenced that does not exist.

This exception class is provided for Delphi compatibility, but is not actually used in FPC.

See also: Exception (1839), EFileNotFoundException (1831), EPathNotFoundException (1837), EPathTooLongException (1837)

76.34 EDivByZero

76.34.1 Description

`EDivByZero` is used when the operating system or CPU signals a division by zero error.

76.35 EEncodingError

76.35.1 Description

`EEncodingError` is the exception classed used by the `TEncoding` class to indicate errors.

See also: TEncoding (1869)

76.36 EExternal

76.36.1 Description

`EExternal` is the base exception for all external exceptions, as reported by the CPU or operating system, as opposed to internal exceptions, which are raised by the program itself. The SysUtils unit converts all operating system errors to descendents of `EExternal`.

See also: EInterror (1833), EExternal (1831), EMathError (1834), EExternalException (1831), EAccessViolation (1829), EPrivilege (1837), EStackOverflow (1838), EControlC (1830)

76.37 EExternalException

76.37.1 Description

`EExternalException` is raised when an external routine raises an exception.

See also: EExternal (1831)

76.38 EFileNotFoundException

76.38.1 Description

`EFileNotFoundException` is an exception raised when a file is referenced that does not exist.

This exception class is provided for Delphi compatibility, but is not actually used in FPC.

See also: Exception (1839), EPathNotFoundException (1837), EDirectoryNotFoundException (1830), EPathTooLongException (1837)

76.39 EFormatError

76.39.1 Description

`EformatError` is raised in case of an error in one of the various Format (1735) functions.

See also: Format (1735)

76.40 EHeapMemoryError

76.40.1 Description

`EHeapMemoryError` is raised when an error occurs in heap (dynamically allocated) memory.

See also: EHeapException (1651), EoutOfMemory (1836), EInvalidPointer (1834)

76.40.2 Method overview

Page	Method	Description
1832	FreeInstance	Free the exception instance.

76.40.3 EHeapMemoryError.FreeInstance

Synopsis: Free the exception instance.

Declaration: `procedure FreeInstance;  Override`

Visibility: public

Description: `FreeInstance` checks whether the exception instance may be freed prior to calling the inherited `FreeInstance`. The exception is only freed in case of normal program shutdown, if a heap error occurred, the exception instance is not freed.

76.41 EInOutArgumentException

76.41.1 Method overview

Page	Method	Description
1832	Create	
1833	CreateFmt	
1832	CreateRes	

76.41.2 EInOutArgumentException.Create

Declaration: `constructor Create(const aMsg: string; const aPath: string);  Overload`

Visibility: public

76.41.3 EInOutArgumentException.CreateRes

Declaration: `constructor CreateRes(ResStringRec: PResStringRec; const aPath: string) ;  Overload`

Visibility: public

76.41.4 EInOutArgumentException.CreateFmt

Declaration: `constructor CreateFmt(const fmt: string; const args: Array of const; const aPath: string); Overload`

Visibility: public

76.42 EInOutError

76.42.1 Description

`EInOutError` is raised when a IO routine of Free Pascal returns an error. The error is converted to an `EInOutError` only if the input/output checking feature of FPC is turned on. The error code of the input/output operation is returned in ErrorCode (**??**).

See also: EInOutError.ErrorCode (**??**)

76.43 EIntError

76.43.1 Description

`EInterror` is used when the operating system or CPU signals an integer operation error, e.g., an overflow.

76.44 EIntfCastError

76.44.1 Description

`EIntfCastError` is raised when an invalid interface cast is encountered.

See also: EInvalidCast (1833)

76.45 EIntOverflow

76.45.1 Description

`EIntOverflow` is used when the operating system or CPU signals a integer overflow error.

See also: EIntError (1833), EDivByZero (1831), ERangeError (1838)

76.46 EInvalidCast

76.46.1 Description

`EInvalidCast` is raised when an invalid typecast error (using the `as` operator) is encountered.

See also: EIntfCastError (1833)

76.47 EInvalidContainer

76.47.1 Description

`EInvalidContainer` is not yet used by Free Pascal, and is provided for Delphi compatibility only.

76.48 EInvalidInsert

76.48.1 Description

`EInvalidInsert` is not yet used by Free Pascal, and is provided for Delphi compatibility only.

76.49 EInvalidOp

76.49.1 Description

`EInvalidOp` is raised when an invalid operation is encountered.

76.50 EInvalidOpException

76.50.1 Description

`EInvalidOpException` is raised when an invalid operation is attempted.

See also: Exception (1839)

76.51 EInvalidPointer

76.51.1 Description

`EInvalidPointer` is raised when an invalid heap pointer is used.

See also: EHeapException (1651), EHeapMemoryError (1832), EOutOfMemory (1836)

76.52 EListError

76.53 EMathError

76.53.1 Description

`EMathError` is used when the operating system or CPU signals a floating point overflow error.

See also: EIntError (1833), EIntOverflow (1833), EDivByZero (1831), ERangeError (1838)

76.54 EMonitor

76.55 EMonitorLockException

76.56 ENoConstructException

76.56.1 Description

`ENoConstructException` is the exception raised when an instance of type `TCharacter` is being created. The TCharacter class only contains static methods, no instances of this class should be instantiated.

76.57 ENoDynLibsSupport

76.57.1 Description

`ENoDynLibsSupport` is raised when no dynamic library support is detected and a dynamic library loading operation is attempted.

See also: EThreadError (1838), Exception (1839)

76.58 ENoMonitorSupportException

76.59 ENoThreadSupport

76.59.1 Description

`ENoThreadSupport` is raised when some thread routines are invoked, and thread support was not enabled when the program was compiled.

76.60 ENotImplemented

76.60.1 Description

`ENotImplemented` can be used to raise an exception when a particular call had been defined, but was not implemented.

76.61 ENotSupportedException

76.61.1 Description

`ENotSupportedException` is an exception raised when a function or procedure is not supported for a certain platform.

This exception class is provided for Delphi compatibility, but is not actually used in FPC.

See also: Exception (1839)

76.62 ENoWideStringSupport

76.62.1 Description

`ENoWideStringSupport` is the exception raised when a run-time 233 occurs, i.e. when widestring routines are called and the application does not contain widestring support.

76.63 EObjectCheck

76.63.1 Description

`EObjectCheck` is raised when the `-CR` (check object references) command-line option or `{$OBJECTCHECKS ON}` directive is in effect and a `Nil` reference to an object or class was encountered.

See also: EAccessViolation (1829)

76.64 EObjectDisposed

76.65 EOperationCancelled

76.66 EOSError

76.66.1 Description

`EOSError` is raised when some Operating System call fails. The ErrorCode (**??**) property contains the operating system error code.

See also: EOSError.ErrorCode (**??**)

76.67 EOutOfMemory

76.67.1 Description

`EOutOfMemory` occurs when memory can no longer be allocated on the heap. An instance of `EOutOfMemory` is allocated on the heap at program startup, so it is available when needed.

See also: EHeapException (1651), EHeapMemoryError (1832), EInvalidPointer (1834)

76.68 EOverflow

76.68.1 Description

`EOverflow` occurs when a float operation overflows. (i.e. result is too big to represent).

See also: EIntError (1833), EIntOverflow (1833), EDivByZero (1831), ERangeError (1838), EUnderFlow (1838)

76.69 EPackageError

76.69.1 Description

`EPackageError` is not yet used by Free Pascal, and is provided for Delphi compatibility only.

76.70 EPathNotFoundException

76.70.1 Description

`EPathNotFoundException` is an exception raised when a path is referenced that does not exist.

This exception class is provided for Delphi compatibility, but is not actually used in FPC.

See also: Exception (1839), EFileNotFoundException (1831), EDirectoryNotFoundException (1830), EPathTooLongException (1837)

76.71 EPathTooLongException

76.71.1 Description

`EPathTooLongException` is an exception raised when a pathname argument is longer than the allowed pathname length for files.

This exception class is provided for Delphi compatibility, but is not actually used in FPC.

See also: Exception (1839), EFileNotFoundException (1831), EDirectoryNotFoundException (1830), EPathNotFoundException (1837)

76.72 EPrivilege

76.72.1 Description

`EPrivilege` is raised when the OS reports that an invalid instruction was executed.

76.73 EProgrammerNotFound

76.73.1 Description

`EProgrammerNotFound` is a Delphi-compatibility exception. It is not used in FPC. Some replies on stackoverflow suggest it is an internal joke of the Delpi team.

76.74 EPropReadOnly

76.74.1 Description

`EPropReadOnly` is raised when an attempt is made to write to a read-only property.

76.75 EPropWriteOnly

76.75.1 Description

`EPropWriteOnly` is raised when an attempt is made to read from a write-only property.

See also: EPropReadOnly (1837)

76.76 ERangeError

76.76.1 Description

`ERangeError` is raised by the Free Pascal runtime library if range checking is on, and a range check error occurs.

See also: EIntError (1833), EDivByZero (1831), EIntOverflow (1833)

76.77 ESafecallException

76.77.1 Description

`ESafecallException` is not yet used by Free Pascal, and is provided for Delphi compatibility only.

76.78 ESigQuit

76.78.1 Description

`ESigQuit` is used when a QUIT signal is received and a run-time error 233 is converted to an exception.

See also: Exception (1839)

76.79 EStackOverflow

76.79.1 Description

`EStackOverflow` occurs when the stack has grown too big (e.g. by infinite recursion).

76.80 EThreadError

76.81 EUnderflow

76.81.1 Description

`EOverflow` occurs when a float operation underflows (i.e. result is too small to represent).

See also: EIntError (1833), EIntOverflow (1833), EDivByZero (1831), ERangeError (1838), EOverFlow (1836)

76.82 EVariantError

76.82.1 Description

`EVariantError` is raised by the internal variant routines.

76.82.2 Method overview

Page	Method	Description
1839	CreateCode	Create an instance of `EVariantError` with a particular error code.

76.82.3 EVariantError.CreateCode

Synopsis: Create an instance of `EVariantError` with a particular error code.

Declaration: `constructor CreateCode(Code: LongInt)`

Visibility: default

Description: `CreateCode` calls the inherited constructor, and sets the ErrCode (**??**) property to `Code`.

See also: ErrCode (**??**)

76.83 Exception

76.83.1 Description

`Exception` is the base class for all exception handling routines in the RTL and FCL. While it is possible to raise an exception with any class descending from `TObject`, it is recommended to use `Exception` as the basis of exception class objects: the `Exception` class introduces properties to associate a message and a help context with the exception being raised. What is more, the SysUtils unit sets the necessary hooks to catch and display unhandled exceptions: in such cases, the message displayed to the end user, will be the message stored in the exception class.

See also: ExceptObject (1706), ExceptAddr (1705), ExceptionErrorMessage (1706), ShowException (1770), Abort (1668)

76.83.2 Method overview

Page	Method	Description
1840	Create	Constructs a new exception object with a given message.
1840	CreateFmt	Constructs a new exception object and formats a new message.
1841	CreateFmtHelp	Constructs a new exception object and sets the help context and formats the message.
1841	CreateHelp	Constructs a new exception object and sets the help context.
1840	CreateRes	Constructs a new exception object and gets the message from a resource.
1841	CreateResFmt	Constructs a new exception object and formats the message from a resource.
1842	CreateResFmtHelp	Constructs a new exception object and sets the help context and formats the message from a resource.
1841	CreateResHelp	Constructs a new exception object and sets the help context and gets the message from a resource.
1842	GetBaseException	
1842	ToString	Nicely formatted version of the exception message.

76.83.3 Property overview

Page	Properties	Access	Description
1842	HelpContext	rw	Help context associated with the exception.
1842	Message	rw	Message associated with the exception.

76.83.4 Exception.Create

Synopsis: Constructs a new exception object with a given message.

Declaration: `constructor Create(const msg: string)`

Visibility: public

Errors: Construction may fail if there is not enough memory on the heap.

See also: Exception.CreateFmt (1840), Exception.Message (1842)

76.83.5 Exception.CreateFmt

Synopsis: Constructs a new exception object and formats a new message.

Declaration: `constructor CreateFmt(const msg: string; const args: Array of const)`

Visibility: public

Errors: Construction may fail if there is not enough memory on the heap.

See also: Exception.Create (1840), Exception.Message (1842), Format (1735)

76.83.6 Exception.CreateRes

Synopsis: Constructs a new exception object and gets the message from a resource.

Declaration: `constructor CreateRes(ResString: PString)`

Visibility: public

Errors: Construction may fail if there is not enough memory on the heap.

See also: Exception.Create (1840), Exception.CreateFmt (1840), Exception.CreateResFmt (1841), Exception.Message (1842)

76.83.7 Exception.CreateResFmt

Synopsis: Constructs a new exception object and formats the message from a resource.

Declaration:
```
constructor CreateResFmt(ResString: PString; const Args: Array of const)
```

Visibility: public

Description: `CreateResFmt` does the same as CreateFmt (1840), but fetches the message from the resource string `ResString`.

Errors: Construction may fail if there is not enough memory on the heap.

See also: Exception.Create (1840), Exception.CreateFmt (1840), Exception.CreateRes (1840), Exception.Message (1842)

76.83.8 Exception.CreateHelp

Synopsis: Constructs a new exception object and sets the help context.

Declaration:
```
constructor CreateHelp(const Msg: string; AHelpContext: LongInt)
```

Visibility: public

Description: `CreateHelp` does the same as the Create (1840) constructor, but additionally stores `AHelpContext` in the HelpContext (1842) property.

See also: Exception.Create (1840)

76.83.9 Exception.CreateFmtHelp

Synopsis: Constructs a new exception object and sets the help context and formats the message.

Declaration:
```
constructor CreateFmtHelp(const Msg: string;
                          const Args: Array of const;
                          AHelpContext: LongInt)
```

Visibility: public

Description: `CreateFmtHelp` does the same as the CreateFmt (1840) constructor, but additionally stores `AHelpContext` in the HelpContext (1842) property.

See also: Exception.CreateFmt (1840)

76.83.10 Exception.CreateResHelp

Synopsis: Constructs a new exception object and sets the help context and gets the message from a resource.

Declaration:
```
constructor CreateResHelp(ResString: PString; AHelpContext: LongInt)
```

Visibility: public

Description: `CreateResHelp` does the same as the CreateRes (1840) constructor, but additionally stores `AHelpContext` in the HelpContext (1842) property.

See also: Exception.CreateRes (1840)

76.83.11 Exception.CreateResFmtHelp

Synopsis: Constructs a new exception object and sets the help context and formats the message from a resource.

Declaration:
```
constructor CreateResFmtHelp(ResString: PString;
                             const Args: Array of const;
                             AHelpContext: LongInt)
```

Visibility: public

Description: `CreateResFmtHelp` does the same as the CreateResFmt (1841) constructor, but additionally stores `AHelpContext` in the HelpContext (1842) property.

See also: Exception.CreateResFmt (1841)

76.83.12 Exception.ToString

Synopsis: Nicely formatted version of the exception message.

Declaration:
```
function ToString : string;  Override
```

Visibility: public

Description: `ToString` overrides the `ToString` method to return a concatenation of classname and Exception.Message (1842).

See also: Exception.Message (1842)

76.83.13 Exception.GetBaseException

Declaration:
```
function GetBaseException : Exception;  Virtual
```

Visibility: public

76.83.14 Exception.HelpContext

Synopsis: Help context associated with the exception.

Declaration:
```
Property HelpContext : LongInt
```

Visibility: public

Access: Read,Write

Description: `HelpContext` is the help context associated with the exception, and can be used to provide context-sensitive help when the exception error message is displayed. It should be set in the exception constructor.

See also: Exception.CreateHelp (1841), Exception.Message (1842)

76.83.15 Exception.Message

Synopsis: Message associated with the exception.

Declaration:
```
Property Message : string
```

Visibility: public

Access: Read,Write

Description: `Message` provides additional information about the exception. It is shown to the user in e.g. the ShowException (1770) routine, and should be set in the constructor when the exception is raised.

See also: Exception.Create (1840), Exception.HelpContext (1842)

76.84 EZeroDivide

76.84.1 Description

`EZeroDivide` occurs when a float division by zero occurs.

See also: EIntError (1833), EIntOverflow (1833), EDivByZero (1831), ERangeError (1838)

76.85 IReadWriteSync

76.85.1 Description

`IReadWriteSync` is an interface for synchronizing read/write operations. Writers are always guaranteed to have exclusive access: readers may or may not have simultaneous access, depending on the implementation.

76.85.2 Method overview

Page	Method	Description
1843	BeginRead	Start a read operation.
1844	BeginWrite	Start a write operation.
1843	EndRead	End a read operation.
1844	EndWrite	End a write operation.

76.85.3 IReadWriteSync.BeginRead

Synopsis: Start a read operation.

Declaration: `procedure BeginRead`

Visibility: default

Description: `BeginRead` indicates that a read operation is about to be started. If a write operation is in progress, then the call will block until the write operation finished. Depending on the implementation the call may also block if another read operation is in progress.

After `BeginRead`, any write operation started with BeginWrite (1844) will block until EndRead (1843) is called.

See also: IReadWriteSync.EndRead (1843), IReadWriteSync.BeginWrite (1844), IReadWriteSync.EndWrite (1844)

76.85.4 IReadWriteSync.EndRead

Synopsis: End a read operation.

Declaration: `procedure EndRead`

Visibility: default

Description: `EndRead` signals the end of a read operation. If there was any blocked write operation, that will be unblocked by a call to `EndRead`.

See also: IReadWriteSync.BeginRead (1843), IReadWriteSync.BeginWrite (1844), IReadWriteSync.EndWrite (1844)

76.85.5 IReadWriteSync.BeginWrite

Synopsis: Start a write operation.

Declaration: `function BeginWrite : Boolean`

Visibility: default

Description: `BeginWrite` signals the begin of a write operation. This call will block if any other read or write operation is currently in progress. It will resume only after all other read or write operations have finished.

See also: IReadWriteSync.EndRead (1843), IReadWriteSync.EndWrite (1844), IReadWriteSync.BeginRead (1843)

76.85.6 IReadWriteSync.EndWrite

Synopsis: End a write operation.

Declaration: `procedure EndWrite`

Visibility: default

Description: `EndWrite` signals the end of a write operation. After the call to `EndWrite` any other read or write operations can start.

See also: IReadWriteSync.EndRead (1843), IReadWriteSync.EndWrite (1844), IReadWriteSync.BeginRead (1843)

76.86 TANSISTRINGBUILDER

76.86.1 Description

`TAnsiStringBuilder` is a class to construct an ansistring out of other strings or characters. It's methods can be used instead of constructing the string manually using the normal + or concat operators. If used properly (e.g. by setting a sufficiently large Capacity (1850)), will result in faster operation. It offers various methods to append to the string.

To create a unicode string, use the TUnicodeStringBuilder (1949) class instead.

See also: TUnicodeStringBuilder (1949)

76.86.2 Method overview

Page	Method	Description
1845	Append	Append data in string form to the buffer.
1846	AppendFormat	Append the result of a `Format`.
1847	AppendLine	Append a string followed by a newline.
1847	Clear	Clear the string being built.
1847	CopyTo	Copy string bytes to an array.
1845	Create	Create a new `TAnsiStringBuilder` instance.
1847	EnsureCapacity	Set the capacity if needed.
1848	Equals	Check if 2 stringbuilders have the same content string.
1848	Insert	Insert data in string form into the buffer at a given position.
1849	Remove	Remove data from the string.
1849	Replace	Replace a range of characters.
1850	ToString	Return the string buffer as an AnsiString.

76.86.3 Property overview

Page	Properties	Access	Description
1850	Capacity	rw	Current capacity of the string buffer.
1850	Chars	rw	Indexed access to the characters in the string.
1850	Length	rw	Currentl length of the string buffer.
1851	MaxCapacity	r	Maximum capacity of the string buffer.

76.86.4 TANSISTRINGBUILDER.Create

Synopsis: Create a new `TAnsiStringBuilder` instance.

Declaration:
```
constructor Create
constructor Create(aCapacity: Integer)
constructor Create(const AValue: ANSISTRING)
constructor Create(aCapacity: Integer; aMaxCapacity: Integer)
constructor Create(const AValue: ANSISTRING; aCapacity: Integer)
constructor Create(const AValue: ANSISTRING; StartIndex: Integer;
                   aLength: Integer; aCapacity: Integer)
```

Visibility: public

Description: `Create` creates an empty `TAnsiStringBuilder` instance. It can optionally set the initial capacity to `aCapacity` and the maximum capacity to `aMaxCapacity` and can also initialize the string buffer with characters from `aValue`: `aLength` characters starting at `StartIndex` (1-based) will be copied to the string buffer. If `aLength` and `StartIndex` are omitted, then all characters are copied.

See also: Capacity (1850), MaxCapacity (1851)

76.86.5 TANSISTRINGBUILDER.Append

Synopsis: Append data in string form to the buffer.

Declaration:
```
function Append(const AValue: Boolean) : TANSISTRINGBUILDER
function Append(const AValue: Byte) : TANSISTRINGBUILDER
function Append(const AValue: AnsiChar) : TANSISTRINGBUILDER
function Append(const AValue: Currency) : TANSISTRINGBUILDER
function Append(const AValue: Double) : TANSISTRINGBUILDER
```

```
function Append(const AValue: SmallInt) : TANSISTRINGBUILDER
function Append(const AValue: LongInt) : TANSISTRINGBUILDER
function Append(const AValue: Int64) : TANSISTRINGBUILDER
function Append(const AValue: TObject) : TANSISTRINGBUILDER
function Append(const AValue: ShortInt) : TANSISTRINGBUILDER
function Append(const AValue: Single) : TANSISTRINGBUILDER
function Append(const AValue: UInt64) : TANSISTRINGBUILDER
function Append(const AValue: Array of AnsiChar) : TANSISTRINGBUILDER
function Append(const AValue: Word) : TANSISTRINGBUILDER
function Append(const AValue: Cardinal) : TANSISTRINGBUILDER
function Append(const AValue: PAnsiChar) : TANSISTRINGBUILDER
function Append(const AValue: RawByteString) : TANSISTRINGBUILDER
function Append(const AValue: AnsiChar; RepeatCount: Integer)
                : TANSISTRINGBUILDER
function Append(const AValue: Array of AnsiChar; StartIndex: Integer;
               SBCharCount: Integer) : TANSISTRINGBUILDER
function Append(const AValue: ANSISTRING; StartIndex: Integer;
               Count: Integer) : TANSISTRINGBUILDER
function Append(const Fmt: ANSISTRING; const Args: Array of const)
                : TANSISTRINGBUILDER
```

Visibility: public

Description: `Append` in its various overloaded forms will append the `aValue` argument to the buffer, after converting it to a string value if necessary:

- For integer values the various IntToSTr (1758) functions will be used.
- For TObject (1635) the TObject.ToString (1635) is used.
- The `repeatCount` argument will repeat the character as many times as instructed using StringOfChar (1635).
- If the `StartIndex` and `SBCharCount` are specified, then `SBCharCount` characters are copied starting from position `StartIndex`.

An overloaded version exists with the same arguments as AppendFormat (1846).

See also: StringOfChar (1635), AppendFormat (1846), AppendLine (1847), Insert (1848), Remove (1849)

76.86.6 TANSISTRINGBUILDER.AppendFormat

Synopsis: Append the result of a `Format`.

Declaration:
```
function AppendFormat(const Fmt: ANSISTRING; const Args: Array of const)
                      : TANSISTRINGBUILDER
```

Visibility: public

Description: `AppendFormat` calls Format (1735) with `fmt` and `Args` and calls append (1845) with the resulting string.

Errors: the call to `Format` may result in an exception if the format parameters do not match the passed arguments.

See also: Append (1845), Insert (1848), Remove (1849), Format (1735)

76.86.7 TANSISTRINGBUILDER.AppendLine

Synopsis: Append a string followed by a newline.

Declaration:
```
function AppendLine : TANSISTRINGBUILDER
function AppendLine(const AValue: RawByteString) : TANSISTRINGBUILDER
```

Visibility: public

Description: `AppendLine` will append `aValue` (if it is specified) and appends the OS linebreak character (sLineBreak (1388)).

See also: TAnsiStringBuilder.AppendFormat (1846), TAnsiStringBuilder.Append (1845), #rtl.system.sLineBreak (1388), Remove (1849), Insert (1848)

76.86.8 TANSISTRINGBUILDER.Clear

Synopsis: Clear the string being built.

Declaration:
```
procedure Clear
```

Visibility: public

Description: `Clear` sets the length of the string to 0 and sets the Capacity (1850) to the default capacity (64 bytes).

See also: Capacity (1850), Remove (1849)

76.86.9 TANSISTRINGBUILDER.CopyTo

Synopsis: Copy string bytes to an array.

Declaration:
```
procedure CopyTo(SourceIndex: Integer;
                 var Destination: Array of AnsiChar;
                 DestinationIndex: Integer; Count: Integer)
```

Visibility: public

Description: `CopyTo` copies `aCount` characters from the string, starting at position `SourceIndex` (0-based) in the string to build, to the character array `Destination` at position `DestinationIndex`.

Errors: If the range of characters to copy falls outside the allowed range of characters in either source (available characters) or destination (available room for characters), then an ERangeError (1838) exception is raised.

See also: ERangeError (1838)

76.86.10 TANSISTRINGBUILDER.EnsureCapacity

Synopsis: Set the capacity if needed.

Declaration:
```
function EnsureCapacity(aCapacity: Integer) : Integer
```

Visibility: public

Description: `EnsureCapacity` checks that `aCapacity` is in the allowed range (less than MaxCapacity (1851)) and if the current capacity is less than `aCapacity`, increases the capacity to `aCapacity`.

Errors: If `aCapacity` is larger than MaxCapacity (1851) or negative, an ERangeError (1838) exception is raised.

See also: MaxCapacity (1851)

76.86.11 TANSISTRINGBUILDER.Equals

Synopsis: Check if 2 stringbuilders have the same content string.

Declaration:
```
function Equals(StringBuilder: TANSISTRINGBUILDER) : Boolean
                ;  Reintroduce
```

Visibility: public

Description: `Equals` checks that the self instance has the same content as the `StringBuilder` instance. It checks the length and content (as bytes) of the strings.

See also: TAnsiStringBuilder.Clear (1847)

76.86.12 TANSISTRINGBUILDER.Insert

Synopsis: Insert data in string form into the buffer at a given position.

Declaration:
```
function Insert(Index: Integer; const AValue: Boolean)
                 : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: Byte) : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: AnsiChar)
                 : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: Currency)
                 : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: Double)
                 : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: SmallInt)
                 : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: LongInt)
                 : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: Array of AnsiChar)
                 : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: Int64)
                 : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: TObject)
                 : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: ShortInt)
                 : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: Single)
                 : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: ANSISTRING)
                 : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: Word) : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: Cardinal)
                 : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: UInt64)
                 : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: ANSISTRING;
                const aRepeatCount: Integer) : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: Array of AnsiChar;
                startIndex: Integer; SBCharCount: Integer)
                 : TANSISTRINGBUILDER
```

Visibility: public

Description: `Insert` in its various overloaded forms will insert the `aValue` argument into the buffer at position `Index` (0-based), after converting it to a string value if necessary:

- For integer values the various IntToSTr (1758) functions will be used.
- For TObject (1635) the TObject.ToString (1635) is used.
- The `repeatCount` argument will insert the character as many times as instructed.
- If the `StartIndex` and `SBCharCount` are specified, then `SBCharCount` characters are copied starting from position `StartIndex`.

See also: AppendFormat (1846), AppendLine (1847), Append (1845), Remove (1849), Replace (1849)

76.86.13 TANSISTRINGBUILDER.Remove

Synopsis: Remove data from the string.

Declaration:
```
function Remove(StartIndex: Integer; RemLength: Integer)
                    : TANSISTRINGBUILDER
```

Visibility: public

Description: `Remove` will remove `RemLength` characters from the string buffer, starting at position `StartIndex` (0-based).

Errors: If the range of characters to remove falls outside the allowed range of characters, then an ERangeError (1838) exception is raised.

See also: AppendFormat (1846), AppendLine (1847), Append (1845), Insert (1848), Clear (1847)

76.86.14 TANSISTRINGBUILDER.Replace

Synopsis: Replace a range of characters.

Declaration:
```
function Replace(const OldChar: AnsiChar; const NewChar: AnsiChar)
                  : TANSISTRINGBUILDER
function Replace(const OldChar: AnsiChar; const NewChar: AnsiChar;
                 StartIndex: Integer; Count: Integer)
                  : TANSISTRINGBUILDER
function Replace(const OldValue: RawByteString;
                 const NewValue: RawByteString) : TANSISTRINGBUILDER
function Replace(const OldValue: RawByteString;
                 const NewValue: RawByteString; StartIndex: Integer;
                 Count: Integer) : TANSISTRINGBUILDER
```

Visibility: public

Description: `Replace` replaces occurrences of `OldValue` with the characters in `NewValue`. The search operation starts at position `StartIndex` (0-based, default 0) till position `StartIndex+aCount` is reached.

Errors: If `StartIndex` or `StartIndex+aCount` is out of range, an ERangeError (1838) exception is raised.

See also: AppendFormat (1846), AppendLine (1847), Append (1845), Insert (1848), Clear (1847), Remove (1849)

76.86.15 TANSISTRINGBUILDER.ToString

Synopsis: Return the string buffer as an AnsiString.

Declaration:
```
function ToString : ANSISTRING;  Override
function ToString(aStartIndex: Integer; aLength: Integer) : ANSISTRING
                  ;  Reintroduce
```

Visibility: public

Description: `ToString` returns the current content of the buffer as an AnsiString.

See also: Clear (1847), CopyTo (1847)

76.86.16 TANSISTRINGBUILDER.Chars

Synopsis: Indexed access to the characters in the string.

Declaration:
```
Property Chars[index: Integer]: AnsiChar; default
```

Visibility: public

Access: Read,Write

Description: `Chars` allows indexed access to the characters in the string using a zero-based `Index`. The characters can be read or written. The allowed range for `Index` is 0 to Length (1850)-1.

See also: Length (1850)

76.86.17 TANSISTRINGBUILDER.Length

Synopsis: Currentl length of the string buffer.

Declaration:
```
Property &Length : Integer
```

Visibility: public

Access: Read,Write

Description: `Length` can be used to get or set the current length (in characters) of the string buffer. When setting the length, the new length is checked whether it lies within the maximum capacity (MaxCapacity (1851)) for the string buffer: if not, an ERangeError (1838) exception is raised. The capacity of the buffer will be adjusted so the buffer has anough capacity to accomodate the new length.

See also: MaxCapacity (1851), Capacity (1850), ERangeError (1838)

76.86.18 TANSISTRINGBUILDER.Capacity

Synopsis: Current capacity of the string buffer.

Declaration:
```
Property Capacity : Integer
```

Visibility: public

Access: Read,Write

Description: `Capacity` can be used to get or set the capacity of the string buffer (in characters). When setting the capacity, the new length is checked whether it lies within the maximum capacity (MaxCapacity (1851)) for the string buffer: if not, an ERangeError (1838) exception is raised.

See also: MaxCapacity (1851), Length (1850), ERangeError (1838)

76.86.19 TANSISTRINGBUILDER.MaxCapacity

Synopsis: Maximum capacity of the string buffer.

Declaration: `Property MaxCapacity : Integer`

Visibility: public

Access: Read

Description: `MaxCapacity` returns the maxmimum capacity of the string buffer (in characters). It is set when the stringbuilder is created, in the constructor of the builder. The Capacity (1850) (and consequently, the Length (1850)) will never be allowed to exceed `MaxCapacity`.

See also: Capacity (1850), Length (1850), ERangeError (1838)

76.87 TBigEndianUnicodeEncoding

76.87.1 Description

`TBigEndianUnicodeEncoding` is the encoding class used to represent the UTF-16 big-endian encoding.

See also: TBigEndianUnicodeEncoding (1851), TUTF7Encoding (1956), TMBCSEncoding (1894), TBigendianUnicodeEncoding (1851)

76.87.2 Method overview

Page	Method	Description
1851	Clone	Clone a `TBigEndianUnicodeEncoding` instance.
1851	GetPreamble	Return BOM marker bytes.

76.87.3 TBigEndianUnicodeEncoding.Clone

Synopsis: Clone a `TBigEndianUnicodeEncoding` instance.

Declaration: `function Clone : TEncoding;  Override`

Visibility: public

Description: `CLone` overrides TEncoding.Clone (1870) to provide a clone of the `TBigEndianUnicodeEncoding` instance.

See also: TEncoding.Clone (1870)

76.87.4 TBigEndianUnicodeEncoding.GetPreamble

Synopsis: Return BOM marker bytes.

Declaration: `function GetPreamble : TBytes;  Override`

Visibility: public

Description: `GetPreamble` overrides TEncoding.GetPreamble (1872) to return the 2 UTF-16 BOM Marker bytes ($FF,$FE).

See also: TEncoding.GetPreamble (1872)

76.88 TBooleanHelper

76.88.1 Description

`TBooleanHelper` is a helper type for the `Boolean` type. It contains mostly conversion routines to and from other types.

See also: TStringHelper (1926), TShortIntHelper (1910), TSmallIntHelper (1922), TWordHelper (1960), TCardinalHelper (1859), TIntegerHelper (1888), TInt64Helper (1884), TQWordHelper (1906), TNativeIntHelper (1898), TByteHelper (1855), TByteBoolHelper (1853), TWordBoolHelper (1959), TLongBoolHelper (1892)

76.88.2 Method overview

Page	Method	Description
1852	Parse	Convert string value to boolean value.
1852	Size	Return the size (in bytes) of the.
1853	ToInteger	Convert to an integer value.
1852	ToString	Convert a boolean value to string.
1853	TryToParse	Try to convert a string to a boolean value.

76.88.3 TBooleanHelper.Parse

Synopsis: Convert string value to boolean value.

Declaration:
```
class function Parse(const S: string) : Boolean;  Static
```

Visibility: public

Description: `Parse` attempts to convert the string `S` to a boolean value. It uses the StrToBool (1787) function to perform the conversion.

Errors: If `S` does not contain a valid string representation, then an EConvertError (1830) exception is raised.

See also: TBooleanHelper.TryToParse (1853), TBooleanHelper.ToString (1852), TBooleanHelper.ToInteger (1853)

76.88.4 TBooleanHelper.Size

Synopsis: Return the size (in bytes) of the.

Declaration:
```
class function Size : Integer;  Static
```

Visibility: public

Description: `Size` returns the size (in bytes) of the boolean value. This is equivalent to `SizeOf(Boolean)`.

See also: SizeOf (1574)

76.88.5 TBooleanHelper.ToString

Synopsis: Convert a boolean value to string.

Declaration:
```
class function ToString(const AValue: Boolean;
                        UseBoolStrs: TUseBoolStrs) : string;  Overload
                        ;  Static
function ToString(UseBoolStrs: TUseBoolStrs) : string;  Overload
```

Visibility: public

Description: `ToString` will, in the class method version, convert the `AValue` boolean to a string representation. In the function method version the boolean value itself (`Self`) will be converted.

If the `UseBoolStrs` parameter equals `TUseBoolStrs.True`, then the string representation will use the boolean strings BoolStrs (1635). The default value for `UseBoolStrs` is `TUseBoolStrs.False`.

The conversion is done using the BoolTostr (1685) function.

See also: BoolStrs (1635), BoolToStr (1685)

76.88.6 TBooleanHelper.TryToParse

Synopsis: Try to convert a string to a boolean value.

Declaration:
```
class function TryToParse(const S: string; out AValue: Boolean)
                          : Boolean;  Static
```

Visibility: public

Description: `TryToParse` will attempt to convert the string `S` to a boolean value. If the attempt is successful, `True` is returned, and the actual value is returned in `AValue`. If the attempt failed, `False` is returned.

See also: TBooleanHelper.Parse (1852), TBooleanHelper.ToString (1852)

76.88.7 TBooleanHelper.ToInteger

Synopsis: Convert to an integer value.

Declaration:
```
function ToInteger : Integer
```

Visibility: public

Description: `ToInteger` will return the boolean value, typecasted to `Integer`.

See also: TBooleanHelper.ToString (1852)

76.89 TByteBoolHelper

76.89.1 Description

`TByteBoolHelper` is a helper type for the `ByteBool` type. It contains mostly conversion routines to and from other types.

See also: TStringHelper (1926), TShortIntHelper (1910), TSmallIntHelper (1922), TWordHelper (1960), TCardinalHelper (1859), TIntegerHelper (1888), TInt64Helper (1884), TQWordHelper (1906), TNativeIntHelper (1898), TByteHelper (1855), TByteBoolHelper (1853), TWordBoolHelper (1959), TLongBoolHelper (1892)

76.89.2 Method overview

Page	Method	Description
1854	Parse	Convert string value to ByteBool value.
1854	Size	Return the size (in bytes) of the.
1855	ToInteger	Convert to an integer value.
1854	ToString	Convert a ByteBool value to string.
1855	TryToParse	Try to convert a string to a ByteBool value.

76.89.3 TByteBoolHelper.Parse

Synopsis: Convert string value to ByteBool value.

Declaration:
```
class function Parse(const S: string) : Boolean;  Static
```

Visibility: public

Description: `Parse` attempts to convert the string `S` to a ByteBool value. It uses the StrToBool (1787) function to perform the conversion.

Errors: If `S` does not contain a valid string representation, then an EConvertError (1830) exception is raised.

See also: TByteBoolHelper.TryToParse (1855), TByteBoolHelper.ToString (1854), TByteBoolHelper.ToInteger (1855)

76.89.4 TByteBoolHelper.Size

Synopsis: Return the size (in bytes) of the.

Declaration:
```
class function Size : Integer;  Static
```

Visibility: public

Description: `Size` returns the size (in bytes) of the ByteBool value. This is equivalent to `SizeOf(ByteBool)`.

See also: SizeOf (1574)

76.89.5 TByteBoolHelper.ToString

Synopsis: Convert a ByteBool value to string.

Declaration:
```
class function ToString(const AValue: Boolean;
                        UseBoolStrs: TUseBoolStrs) : string;  Overload
                        ;  Static
function ToString(UseBoolStrs: TUseBoolStrs) : string;  Overload
```

Visibility: public

Description: `ToString` will, in the class method version, convert the `AValue` ByteBool to a string representation. In the function method version the ByteBool value itself (`Self`) will be converted.

If the `UseBoolStrs` parameter equals `TUseBoolStrs.True`, then the string representation will use the ByteBool strings BoolStrs (1635). The default value for `UseBoolStrs` is `TUseBoolStrs.False`.

The conversion is done using the BoolTostr (1685) function.

See also: BoolStrs (1635), BoolToStr (1685)

76.89.6 TByteBoolHelper.TryToParse

Synopsis: Try to convert a string to a ByteBool value.

Declaration:
```
class function TryToParse(const S: string; out AValue: Boolean)
                          : Boolean;  Static
```

Visibility: public

Description: `TryToParse` will attempt to convert the string `S` to a ByteBool value. If the attempt is successful, `True` is returned, and the actual value is returned in `AValue`. If the attempt failed, `False` is returned.

See also: TByteBoolHelper.Parse (1854), TByteBoolHelper.ToString (1854)

76.89.7 TByteBoolHelper.ToInteger

Synopsis: Convert to an integer value.

Declaration:
```
function ToInteger : Integer
```

Visibility: public

Description: `ToInteger` will return the ByteBool value, typecasted to `Integer`.

See also: TByteBoolHelper.ToString (1854)

76.90 TByteHelper

76.90.1 Description

`TByteHelper` contains some auxiliary routines for a byte-typed ordinal value. It consists mainly of conversion routines to and from other types.

See also: TStringHelper (1926), TShortIntHelper (1910), TSmallIntHelper (1922), TWordHelper (1960), TCardinalHelper (1859), TIntegerHelper (1888), TInt64Helper (1884), TQWordHelper (1906), TNativeIntHelper (1898), TNativeUIntHelper (1902)

76.90.2 Method overview

Page	Method	Description
1858	ClearBit	Set bit to 0.
1856	Parse	Convert from a string.
1858	SetBit	Set bit to 1.
1856	Size	Size, in bytes, of the byte value.
1859	TestBit	Check bit.
1857	ToBinString	Convert to a binary string representation.
1857	ToBoolean	Convert to a boolean value.
1857	ToDouble	Convert to a double-sized floating point value.
1857	ToExtended	Convert to an extended-sized floating point value.
1859	ToggleBit	Invert bit.
1858	ToHexString	Convert to a hexadecimal string representation.
1858	ToSingle	Convert to an single-sized floating point value.
1856	ToString	Convert the value to string.
1856	TryParse	Try to convert a string to a byte, report success or failure.

76.90.3 TByteHelper.Parse

Synopsis: Convert from a string.

Declaration:
```
class function Parse(const AString: string) : Byte;  Static
```

Visibility: public

Description: `Parse` will attempt to convert the string `AString` to a byte value. It uses the StrToInt (1793) function to perform the conversion, so no localization is taken into account.

Errors: If the string does not contain a valid byte value, an EConvertError (1830) exception is raised.

See also: TByteHelper.ToString (1856), TByteHelper.TryParse (1856), StrToInt (1793)

76.90.4 TByteHelper.Size

Synopsis: Size, in bytes, of the byte value.

Declaration:
```
class function Size : Integer;  Static
```

Visibility: public

Description: `Size` returns the size (in bytes) of the byte value. This is equivalent to `SizeOf(Byte)`.

Errors: None.

See also: SizeOf (1574)

76.90.5 TByteHelper.ToString

Synopsis: Convert the value to string.

Declaration:
```
class function ToString(const AValue: Byte) : string;  Overload;  Static
function ToString : string;  Overload
```

Visibility: public

Description: `ToString` will, in the class function variant of this method, convert `AValue` to a string representation. In the regular method overloaded version of `ToString`, the byte value itself is used. The IntToStr (1758) function is used to do the conversion.

See also: TByteHelper.Parse (1856), IntToStr (1758)

76.90.6 TByteHelper.TryParse

Synopsis: Try to convert a string to a byte, report success or failure.

Declaration:
```
class function TryParse(const AString: string; out AValue: Byte)
                        : Boolean;  Static
```

Visibility: public

Description: `TryParse` attempts to convert the string `AString` to a byte, and reports the success of the attempt. If the attempt is successful, then `True` is returned, and the actual value of the byte is returned in `AValue`.

It uses the val (1635) function to perform the conversion, so no localization is taken into account.

See also: TByteHelper.Parse (1856), Val (1598)

76.90.7 TByteHelper.ToBoolean

Synopsis: Convert to a boolean value.

Declaration: `function ToBoolean : Boolean`

Visibility: public

Description: `ToBoolean` converts the byte value to a boolean: it returns `True` if the value is nonzero, `False` if it is zero.

See also: TByteHelper.ToSingle (1858), TByteHelper.ToDouble (1857), TByteHelper.ToExtended (1857), TByteHelper.ToString (1856), TByteHelper.ToHexString (1858)

76.90.8 TByteHelper.ToDouble

Synopsis: Convert to a double-sized floating point value.

Declaration: `function ToDouble : Double`

Visibility: public

Description: `ToDouble` converts the byte value to a double-sized floating point value.

See also: TByteHelper.ToBoolean (1857), TByteHelper.ToExtended (1857), TByteHelper.ToSingle (1858), TByteHelper.ToString (1856), TByteHelper.ToHexString (1858)

76.90.9 TByteHelper.ToExtended

Synopsis: Convert to an extended-sized floating point value.

Declaration: `function ToExtended : Extended`

Visibility: public

Description: `ToDouble` converts the byte value to an extended-sized floating point value

See also: TByteHelper.ToBoolean (1857), TByteHelper.ToSingle (1858), TByteHelper.ToDouble (1857), TByteHelper.ToString (1856), TByteHelper.ToHexString (1858)

76.90.10 TByteHelper.ToBinString

Synopsis: Convert to a binary string representation.

Declaration: `function ToBinString : string`

Visibility: public

Description: `ToBinString` converts the byte value to a binary string representation, showing all 8 bits.

See also: TByteHelper.ToHexString (1858), TByteHelper.ToString (1856)

76.90.11 TByteHelper.ToHexString

Synopsis: Convert to a hexadecimal string representation.

Declaration:
```
function ToHexString(const AMinDigits: Integer) : string;  Overload
function ToHexString : string;  Overload
```

Visibility: public

Description: `ToHexString` converts the byte value to a hexadecimal string representation. The `AMinDigits` argument specifies the minimal number of characters in the resulting string. The string will be left-padded with zeroes if the representation contains less than `AMinDigits` characters.

See also: TByteHelper.ToBoolean (1857), TByteHelper.ToSingle (1858), TByteHelper.ToDouble (1857), TByteHelper.ToString (1856), TByteHelper.ToExtended (1857)

76.90.12 TByteHelper.ToSingle

Synopsis: Convert to an single-sized floating point value.

Declaration:
```
function ToSingle : Single
```

Visibility: public

Description: `ToSingle` converts the byte value to a single-sized floating point value.

See also: TByteHelper.ToBoolean (1857), TByteHelper.ToDouble (1857), TByteHelper.ToExtended (1857), TByteHelper.ToString (1856), TByteHelper.ToHexString (1858)

76.90.13 TByteHelper.SetBit

Synopsis: Set bit to 1.

Declaration:
```
function SetBit(const Index: TByteBitIndex) : Byte
```

Visibility: public

Description: `SetBit` puts value 1 to bit number determined by argument `Index`.

See also: TByteHelper.ClearBit (1858), TByteHelper.ToggleBit (1859), TByteHelper.TestBit (1859), TByteHelper.HighestSetBitPos (1855), TByteHelper.LowestSetBitPos (1855), TByteHelper.SetBitsCount (1855), TByteHelper.Bits (1855)

76.90.14 TByteHelper.ClearBit

Synopsis: Set bit to 0.

Declaration:
```
function ClearBit(const Index: TByteBitIndex) : Byte
```

Visibility: public

Description: `ClearBit` puts value 0 to bit number determined by argument `Index`.

See also: TByteHelper.SetBit (1858), TByteHelper.ToggleBit (1859), TByteHelper.TestBit (1859), TByteHelper.HighestSetBitPos (1855), TByteHelper.LowestSetBitPos (1855), TByteHelper.SetBitsCount (1855), TByteHelper.Bits (1855)

76.90.15 TByteHelper.ToggleBit

Synopsis: Invert bit.

Declaration: `function ToggleBit(const Index: TByteBitIndex) : Byte`

Visibility: public

Description: `ToggleBit` reads bit value from bit number determined by argument `Index`, inverts it (if it was 0 it becomes 1, and vice versa), and stores it back.

See also: TByteHelper.SetBit (1858), TByteHelper.ClearBit (1858), TByteHelper.TestBit (1859), TByteHelper.HighestSetBitPos (1855), TByteHelper.LowestSetBitPos (1855), TByteHelper.SetBitsCount (1855), TByteHelper.Bits (1855)

76.90.16 TByteHelper.TestBit

Synopsis: Check bit.

Declaration: `function TestBit(const Index: TByteBitIndex) : Boolean`

Visibility: public

Description: `TestBit` reads boolean value (true if bit is 1, and false if bit is 0) from bit number determined by argument `Index`.

See also: TByteHelper.SetBit (1858), TByteHelper.ClearBit (1858), TByteHelper.ToggleBit (1859), TByteHelper.HighestSetBitPos (1855), TByteHelper.LowestSetBitPos (1855), TByteHelper.SetBitsCount (1855), TByteHelper.Bits (1855)

76.91 TCardinalHelper

76.91.1 Description

`TCardinalHelper` contains some auxiliary routines for a Cardinal-typed ordinal value. It consists mainly of conversion routines to and from other types.

See also: TStringHelper (1926), TShortIntHelper (1910), TSmallIntHelper (1922), TWordHelper (1960), TByteHelper (1855), TIntegerHelper (1888), TInt64Helper (1884), TQWordHelper (1906), TNativeIntHelper (1898), TNativeUIntHelper (1902)

76.91.2 Method overview

Page	Method	Description
1863	ClearBit	Set bit to 0.
1860	Parse	Convert from a string.
1862	SetBit	Set bit to 1.
1860	Size	Size, in bytes, of the Cardinal value.
1863	TestBit	Check bit.
1862	ToBinString	Convert to a binary string representation.
1861	ToBoolean	Convert to a boolean value.
1861	ToDouble	Convert to a double-sized floating point value.
1861	ToExtended	Convert to an extended-sized floating point value.
1863	ToggleBit	Invert bit.
1862	ToHexString	Convert to a hexadecimal string representation.
1862	ToSingle	Convert to an single-sized floating point value.
1860	ToString	Convert the value to string.
1861	TryParse	Try to convert a string to a Cardinal, report success or failure.

76.91.3 TCardinalHelper.Parse

Synopsis: Convert from a string.

Declaration:
```
class function Parse(const AString: string) : Cardinal;  Static
```

Visibility: public

Description: `Parse` will attempt to convert the string `AString` to a Cardinal value. It uses the StrToInt (1793) function to perform the conversion, so no localization is taken into account.

Errors: If the string does not contain a valid Cardinal value, an EConvertError (1830) exception is raised.

See also: TCardinalHelper.ToString (1860), TCardinalHelper.TryParse (1861), StrToInt (1793)

76.91.4 TCardinalHelper.Size

Synopsis: Size, in bytes, of the Cardinal value.

Declaration:
```
class function Size : Integer;  Static
```

Visibility: public

Description: `Size` returns the size (in Cardinals) of the Cardinal value. This is equivalent to `SizeOf(Cardinal)`.

Errors: None.

See also: SizeOf (1574)

76.91.5 TCardinalHelper.ToString

Synopsis: Convert the value to string.

Declaration:
```
class function ToString(const AValue: Cardinal) : string;  Overload
                              ;  Static
function ToString : string;  Overload
```

Visibility: public

Description: `ToString` will, in the class function variant of this method, convert `AValue` to a string representation. In the regular method overloaded version of `ToString`, the Cardinal value itself is used. The IntToStr (1758) function is used to do the conversion.

See also: TCardinalHelper.Parse (1860), IntToStr (1758)

76.91.6 TCardinalHelper.TryParse

Synopsis: Try to convert a string to a Cardinal, report success or failure.

Declaration:
```
class function TryParse(const AString: string; out AValue: Cardinal)
                        : Boolean;  Static
```

Visibility: public

Description: `TryParse` attempts to convert the string `AString` to a Cardinal, and reports the success of the attempt. If the attempt is successful, then `True` is returned, and the actual value of the Cardinal is returned in `AValue`.

It uses the val (1635) function to perform the conversion, so no localization is taken into account.

See also: TCardinalHelper.Parse (1860), Val (1598)

76.91.7 TCardinalHelper.ToBoolean

Synopsis: Convert to a boolean value.

Declaration:
```
function ToBoolean : Boolean
```

Visibility: public

Description: `ToBoolean` converts the Cardinal value to a boolean: it returns `True` if the value is nonzero, `False` if it is zero.

See also: TCardinalHelper.ToSingle (1862), TCardinalHelper.ToDouble (1861), TCardinalHelper.ToExtended (1861), TCardinalHelper.ToString (1860), TCardinalHelper.ToHexString (1862)

76.91.8 TCardinalHelper.ToDouble

Synopsis: Convert to a double-sized floating point value.

Declaration:
```
function ToDouble : Double
```

Visibility: public

Description: `ToDouble` converts the Cardinal value to a double-sized floating point value.

See also: TCardinalHelper.ToBoolean (1861), TCardinalHelper.ToExtended (1861), TCardinalHelper.ToSingle (1862), TCardinalHelper.ToString (1860), TCardinalHelper.ToHexString (1862)

76.91.9 TCardinalHelper.ToExtended

Synopsis: Convert to an extended-sized floating point value.

Declaration:
```
function ToExtended : Extended
```

Visibility: public

Description: `ToDouble` converts the Cardinal value to an extended-sized floating point value.

See also: TCardinalHelper.ToBoolean (1861), TCardinalHelper.ToSingle (1862), TCardinalHelper.ToDouble (1861), TCardinalHelper.ToString (1860), TCardinalHelper.ToHexString (1862)

76.91.10 TCardinalHelper.ToBinString

Synopsis: Convert to a binary string representation.

Declaration: `function ToBinString : string`

Visibility: public

Description: `ToBinString` converts the Cardinal value to a binary string representation, showing all 32 bits.

See also: TCardinalHelper.ToHexString (1862), TCardinalHelper.ToString (1860)

76.91.11 TCardinalHelper.ToHexString

Synopsis: Convert to a hexadecimal string representation.

Declaration:
```
function ToHexString(const AMinDigits: Integer) : string;  Overload
function ToHexString : string;  Overload
```

Visibility: public

Description: `ToHexString` converts the Cardinal value to a hexadecimal string representation. The `AMinDigits` argument specifies the minimal number of characters in the resulting string. The string will be left-padded with zeroes if the representation contains less than `AMinDigits` characters.

See also: TCardinalHelper.ToBoolean (1861), TCardinalHelper.ToSingle (1862), TCardinalHelper.ToDouble (1861), TCardinalHelper.ToString (1860), TCardinalHelper.ToExtended (1861)

76.91.12 TCardinalHelper.ToSingle

Synopsis: Convert to an single-sized floating point value.

Declaration: `function ToSingle : Single`

Visibility: public

Description: `ToSingle` converts the Cardinal value to a single-sized floating point value.

See also: TCardinalHelper.ToBoolean (1861), TCardinalHelper.ToDouble (1861), TCardinalHelper.ToExtended (1861), TCardinalHelper.ToString (1860), TCardinalHelper.ToHexString (1862)

76.91.13 TCardinalHelper.SetBit

Synopsis: Set bit to 1.

Declaration: `function SetBit(const Index: TCardinalBitIndex) : Cardinal`

Visibility: public

Description: `SetBit` puts value 1 to bit number determined by argument `Index`.

See also: TCardinalHelper.ClearBit (1863), TCardinalHelper.ToggleBit (1863), TCardinalHelper.TestBit (1863), TCardinalHelper.HighestSetBitPos (1859), TCardinalHelper.LowestSetBitPos (1859), TCardinalHelper.SetBitsCount (1859), TCardinalHelper.Bits (1859)

76.91.14 TCardinalHelper.ClearBit

Synopsis: Set bit to 0.

Declaration: `function ClearBit(const Index: TCardinalBitIndex) : Cardinal`

Visibility: public

Description: `ClearBit` puts value 0 to bit number determined by argument `Index`.

See also: TCardinalHelper.SetBit (1862), TCardinalHelper.ToggleBit (1863), TCardinalHelper.TestBit (1863), TCardinalHelper.HighestSetBitPos (1859), TCardinalHelper.LowestSetBitPos (1859), TCardinalHelper.SetBitsCount (1859), TCardinalHelper.Bits (1859)

76.91.15 TCardinalHelper.ToggleBit

Synopsis: Invert bit.

Declaration: `function ToggleBit(const Index: TCardinalBitIndex) : Cardinal`

Visibility: public

Description: `ToggleBit` reads bit value from bit number determined by argument `Index`, inverts it (if it was 0 it becomes 1, and vice versa), and stores it back.

See also: TCardinalHelper.SetBit (1862), TCardinalHelper.ToggleBit (1863), TCardinalHelper.TestBit (1863), TCardinalHelper.HighestSetBitPos (1859), TCardinalHelper.LowestSetBitPos (1859), TCardinalHelper.SetBitsCount (1859), TCardinalHelper.Bits (1859)

76.91.16 TCardinalHelper.TestBit

Synopsis: Check bit.

Declaration: `function TestBit(const Index: TCardinalBitIndex) : Boolean`

Visibility: public

Description: `TestBit` reads boolean value (true if bit is 1, and false if bit is 0) from bit number determined by argument `Index`.

See also: TCardinalHelper.SetBit (1862), TCardinalHelper.ToggleBit (1863), TCardinalHelper.TestBit (1863), TCardinalHelper.HighestSetBitPos (1859), TCardinalHelper.LowestSetBitPos (1859), TCardinalHelper.SetBitsCount (1859), TCardinalHelper.Bits (1859)

76.92 TDoubleHelper

76.92.1 Description

`TDoubleHelper` is the helper type for the Double-sized floating point type. It contains some conversion methods, as well as access to the low-level structure of the floating-point representation of a Double.

See also: TDoubleHelper (1863), TExtendedHelper (1876)

76.92.2 Method overview

Page	Method	Description
1867	BuildUp	Build a Double-sized floating point from its composing parts.
1867	Exponent	Exponent of the floating-point value.
1867	Fraction	Fraction of the floating-point value.
1864	IsInfinity	Check whether a value is positive or negative infinity.
1864	IsNan	Check whether a value equals `NaN`.
1865	IsNegativeInfinity	Check whether a value is negative infinity.
1865	IsPositiveInfinity	Check whether a value is positive infinity.
1868	Mantissa	Mantissa of the floating-point.
1865	Parse	Convert a string to a floating point value.
1866	Size	Size (in bytes) of a Double-sized floating point value.
1868	SpecialType	Return the type of the Double-sized floating point value.
1866	ToString	Convert a Double-sized floating point value to a string.
1867	TryParse	Try to convert a string to a Double-sized floating point value.

76.92.3 Property overview

Page	Properties	Access	Description
1868	Bytes	rw	Indexed access to the individual bytes of the floating point value.
1869	Exp	rw	The bit pattern of the exponent as stored in memory.
1869	Frac	rw	Bitpattern that makes up the fractional part.
1869	Sign	rw	Sign of the floating point value.
1868	Words	rw	Indexed access to the words that make up the floating point value.

76.92.4 TDoubleHelper.IsInfinity

Synopsis: Check whether a value is positive or negative infinity.

Declaration:
```
class function IsInfinity(const AValue: Double) : Boolean;  Overload
                              ;  Static
function IsInfinity : Boolean;  Overload
```

Visibility: public

Description: `IsInfinity` checks whether a Double-sized floating point value represents a positive or negative infinity. If so, it returns `True`. When the class function version is used, the value can be specified using `AValue`. In the method version, the used value is (`Self`).

See also: TDoubleHelper.IsNan (1864), TDoubleHelper.IsPositiveInfinity (1865), TDoubleHelper.IsNegativeInfinity (1865)

76.92.5 TDoubleHelper.IsNan

Synopsis: Check whether a value equals `NaN`.

Declaration:
```
class function IsNan(const AValue: Double) : Boolean;  Overload;  Static
function IsNan : Boolean;  Overload
```

Visibility: public

Description: `IsNan` checks whether a Double-sized floating point value is `NaN` (Not a Number). If so, it returns `True`. When the class function version is used, the value can be specified using `AValue`. In the method version, the used value is (`Self`).

See also: TDoubleHelper.IsInfinity (1864), TDoubleHelper.IsPositiveInfinity (1865), TDoubleHelper.IsNegativeInfinity (1865)

76.92.6 TDoubleHelper.IsNegativeInfinity

Synopsis: Check whether a value is negative infinity.

Declaration:
```
class function IsNegativeInfinity(const AValue: Double) : Boolean
                                  ;  Overload;  Static
function IsNegativeInfinity : Boolean;  Overload
```

Visibility: public

Description: `IsNegativeInfinity` checks whether a Double-sized floating point value represents a negative infinity. If so, it returns `True`. When the class function version is used, the value can be specified using `AValue`. In the method version, the used value is (`Self`).

See also: TDoubleHelper.IsNan (1864), TDoubleHelper.IsPositiveInfinity (1865), TDoubleHelper.IsInfinity (1864)

76.92.7 TDoubleHelper.IsPositiveInfinity

Synopsis: Check whether a value is positive infinity.

Declaration:
```
class function IsPositiveInfinity(const AValue: Double) : Boolean
                                  ;  Overload;  Static
function IsPositiveInfinity : Boolean;  Overload
```

Visibility: public

Description: `IsPositiveInfinity` checks whether a Double-sized floating point value represents a positive infinity. If so, it returns `True`. When the class function version is used, the value can be specified using `AValue`. In the method version, the used value is (`Self`).

See also: TDoubleHelper.IsNan (1864), TDoubleHelper.IsNegativeInfinity (1865), TDoubleHelper.IsInfinity (1864)

76.92.8 TDoubleHelper.Parse

Synopsis: Convert a string to a floating point value.

Declaration:
```
class function Parse(const AString: string) : Double;  Overload;  Static
class function Parse(const AString: string;
                     const AFormatSettings: TFormatSettings) : Double
                     ;  Overload;  Static
```

Visibility: public

Description: `Parse` will try to convert `AString` to a Double-sized floating point value. It will take into account internationalization settings. (it uses `FloatToStr`).

Errors: If the string `AString` is not a valid floating-point value, a EConvertError (1830) exception is raised.

See also: FloatToStr (1729), TDoubleHelper.ToString (1866), TDoubleHelper.TryParse (1867)

76.92.9 TDoubleHelper.Size

Synopsis: Size (in bytes) of a Double-sized floating point value.

Declaration:
```
class function Size : Integer;  Static
```

Visibility: public

Description: `Size` is the size (in bytes) of a Double-sized floating point value. It is equivalent to calling `SizeOf(Double)`.

See also: SizeOf (1574)

76.92.10 TDoubleHelper.ToString

Synopsis: Convert a Double-sized floating point value to a string.

Declaration:
```
class function ToString(const AValue: Double) : string;  Overload
                        ;  Static
class function ToString(const AValue: Double;
                        const AFormat: TFloatFormat;
                        const APrecision: Integer;
                        const ADigits: Integer) : string;  Overload
                        ;  Static
class function ToString(const AValue: Double;
                        const AFormat: TFloatFormat;
                        const APrecision: Integer;
                        const ADigits: Integer;
                        const AFormatSettings: TFormatSettings) : string
                        ;  Overload;  Static
class function ToString(const AValue: Double;
                        const AFormatSettings: TFormatSettings) : string
                        ;  Overload;  Static
function ToString(const AFormat: TFloatFormat;
                  const APrecision: Integer; const ADigits: Integer)
                   : string;  Overload
function ToString(const AFormat: TFloatFormat;
                  const APrecision: Integer; const ADigits: Integer;
                  const AFormatSettings: TFormatSettings) : string
                  ;  Overload
function ToString(const AFormatSettings: TFormatSettings) : string
                  ;  Overload
function ToString : string;  Overload
```

Visibility: public

Description: `ToString` will convert `AValue` (or `Self` in the plain method version) to a string. Optionally `FormatSettings` can be specified, to be able to specify the decimal separator character to use.

Additionally, a precision `APrecision` and number of digits `ADigits` can be specified, in conjunction with a `AFormat` parameter to specify the form in which the floating-point value must be represented. (see TFloatFormat (1656) for an explanation of the various values). In this case, FloatToStrF (1731) is used to format the value. In the absence of these parameters, FloatToStr (1729) is called.

See also: FloatToStr (1729), FloatToStrF (1731), TFloatFormat (1656)

76.92.11 TDoubleHelper.TryParse

Synopsis: Try to convert a string to a Double-sized floating point value.

Declaration:
```
class function TryParse(const AString: string; out AValue: Double)
                        : Boolean;  Overload;  Static
class function TryParse(const AString: string; out AValue: Double;
                        const AFormatSettings: TFormatSettings) : Boolean
                        ;  Overload;  Static
```

Visibility: public

Description: `TryParse` attempts to convert the string `AString` to a Double-sized floating point value and reports `True` if the conversion was successful. In that case the parsed value is returned in `AValue`.

If the conversion failed, `False` is returned.

See also: TDoubleHelper.Parse (1865), TDoubleHelper.ToString (1866)

76.92.12 TDoubleHelper.BuildUp

Synopsis: Build a Double-sized floating point from its composing parts.

Declaration:
```
procedure BuildUp(const ASignFlag: Boolean; const AMantissa: QWord;
                  const AExponent: Integer)
```

Visibility: public

Description: `BuildUp` will compose a Double-sized floating point value from the sign `ASignFlag`, mantissa `AMantissa` and exponent `AExponent`. It simply sets the Sign (1869), Exp (1869) and Frac (1869) properties in 1 call.

See also: TDoubleHelper.Sign (1869), TDoubleHelper.Exp (1869), TDoubleHelper.Frac (1869)

76.92.13 TDoubleHelper.Exponent

Synopsis: Exponent of the floating-point value.

Declaration:
```
function Exponent : Integer
```

Visibility: public

Description: `Exponent` is the value `X` in the representation of the floating-point value in $m*2\hat{X}$, i.e. the exponent.

See also: TDoubleHelper.Sign (1869), TDoubleHelper.Exp (1869), TDoubleHelper.Frac (1869), TDoubleHelper.Fraction (1867), TDoubleHelper.Mantissa (1868)

76.92.14 TDoubleHelper.Fraction

Synopsis: Fraction of the floating-point value.

Declaration:
```
function Fraction : Extended
```

Visibility: public

Description: `Fraction` is the decimal part of the floating-point value.

See also: TDoubleHelper.Sign (1869), TDoubleHelper.Exp (1869), TDoubleHelper.Exponent (1867), TDoubleHelper.Frac (1869), TDoubleHelper.Mantissa (1868)

76.92.15 TDoubleHelper.Mantissa

Synopsis: Mantissa of the floating-point.

Declaration: `function Mantissa : QWord`

Visibility: public

Description: `Mantissa` is the value of the significand without the hidden bit. This means it the plain bit pattern as it is stored in memory.

See also: TDoubleHelper.Sign (1869), TDoubleHelper.Exp (1869), TDoubleHelper.Exponent (1867), TDoubleHelper.Frac (1869), TDoubleHelper.Fraction (1867)

76.92.16 TDoubleHelper.SpecialType

Synopsis: Return the type of the Double-sized floating point value.

Declaration: `function SpecialType : TFloatSpecial`

Visibility: public

Description: `SpecialType` checks whether the Double-sized floating point value equals one of several special values, and returns an enumerated value describing which value this is. See TFloatSpecial (1635) for an explanation of the possible values.

See also: TFloatSpecial (1635)

76.92.17 TDoubleHelper.Bytes

Synopsis: Indexed access to the individual bytes of the floating point value.

Declaration: `Property Bytes[AIndex: Cardinal]: Byte`

Visibility: public

Access: Read,Write

Description: `Bytes` can be used to get or set the various bytes that make up the Double-sized floating point value. The index runs from 0 to `Size-1`.

See also: TDoubleHelper.Words (1868), TDoubleHelper.Size (1866)

76.92.18 TDoubleHelper.Words

Synopsis: Indexed access to the words that make up the floating point value.

Declaration: `Property Words[AIndex: Cardinal]: Word`

Visibility: public

Access: Read,Write

Description: `Words` can be used to get or set the various bytes that make up the Double-sized floating point value. The index runs from 0 to `(Size-1) div 2`.

See also: TDoubleHelper.Bytes (1868), TDoubleHelper.Size (1866)

76.92.19 TDoubleHelper.Sign

Synopsis: Sign of the floating point value.

Declaration: `Property Sign : Boolean`

Visibility: public

Access: Read,Write

Description: `Sign` returns `True` if the sign bit of the value is set (i.e. it is a negative value) or `False` if it is not set (i.e. it is a positive value).

See also: TDoubleHelper.Bytes (1868), TDoubleHelper.Exp (1869), TDoubleHelper.Frac (1869), TDoubleHelper.Mantissa (1868), TDoubleHelper.Fraction (1867), TDoubleHelper.Exponent (1867)

76.92.20 TDoubleHelper.Exp

Synopsis: The bit pattern of the exponent as stored in memory.

Declaration: `Property Exp : QWord`

Visibility: public

Access: Read,Write

Description: `Exp` is the internal representation of the Exponent (1867).

See also: TDoubleHelper.Bytes (1868), TDoubleHelper.Sign (1869), TDoubleHelper.Frac (1869), TDoubleHelper.Mantissa (1868), TDoubleHelper.Fraction (1867), TDoubleHelper.Exponent (1867)

76.92.21 TDoubleHelper.Frac

Synopsis: Bitpattern that makes up the fractional part.

Declaration: `Property Frac : QWord`

Visibility: public

Access: Read,Write

Description: `Frac` is the bit pattern representing the fractional part (significand) including the preceding 1 (the hidden bit).

See also: TDoubleHelper.Bytes (1868), TDoubleHelper.Sign (1869), TDoubleHelper.Exp (1869), TDoubleHelper.Mantissa (1868), TDoubleHelper.Fraction (1867), TDoubleHelper.Exponent (1867)

76.93 TEncoding

76.93.1 Description

`TEncoding` is a mostly abstract class that contains various methods to deal with different encodings in single- and double-byte strings. In practice, one instance of a descendent of this class for each needed encoding can be instantiated and used. The class contains several class methods and properties to create such instances, and has several often-used instances available as class properties, in particular the default encoding.

When using encodings (and in particular TEncoding.Default (1875) or TEncoding.ANSI (1874)), if the DefaultSystemCodePage (1443) changes, you should call TEncoding.FreeEncodings (1869) to regenerate the encodings.

See also: TEncoding.ANSI (1874), TEncoding.ASCII (1874), TEncoding.Default (1875), TEncoding.BigEndianUnicode (1874), TEncoding.Unicode (1875), TEncoding.UTF7 (1875), TEncoding.UTF8 (1876)

76.93.2 Method overview

Page	Method	Description
1870	Clone	Clone a `TEncoding` instance.
1870	Convert	Convert an array of bytes from one encoding to another.
1871	GetBufferEncoding	Attempt to guess the encoding of a buffer.
1872	GetEncoding	Get an encoding instance for a given codepage.
1872	GetMaxByteCount	Returns the maximum number of bytes needed to represent a string.
1872	GetMaxCharCount	Return the maximum number of characters that can be represented in a number of bytes.
1872	GetPreamble	Return the BOM Marker used by the encoding.
1873	GetString	Return a string based on an array of bytes.
1871	IsStandardEncoding	Check if the encoding is one of the standard encodings.

76.93.3 Property overview

Page	Properties	Access	Description
1874	ANSI	r	Ansi encoding instance.
1874	ASCII	r	ASCII encoding instance.
1874	BigEndianUnicode	r	Big-endian Unicode (UTF16BE) encoding instance.
1873	CodePage	r	Codepage for this encoding.
1875	Default	r	Default codepage.
1873	EncodingName	r	Name of this encoding.
1874	IsSingleByte	r	Is the encoding a single-byte encoding or not ?
1875	SystemEncoding	r	System encoding.
1875	Unicode	r	UTF16 encoding instance.
1875	UTF7	r	UTF7 encoding instance.
1876	UTF8	r	UTF8 encoding instance.

76.93.4 TEncoding.Clone

Synopsis: Clone a `TEncoding` instance.

Declaration:
```
function Clone : TEncoding;  Virtual
```

Visibility: public

Description: `Clone` creates a copy of a `TEncoding` instance. This method returns `Nil` in `TEncoding` and must be implemented in descendent classes.

See also: TEncoding.Create (1869)

76.93.5 TEncoding.Convert

Synopsis: Convert an array of bytes from one encoding to another.

Declaration:
```
class function Convert(Source: TEncoding; Destination: TEncoding;
                       const Bytes: TBytes) : TBytes;  Overload
class function Convert(Source: TEncoding; Destination: TEncoding;
```

```
                                const Bytes: TBytes; StartIndex: Integer;
                                Count: Integer) : TBytes;  Overload
```

Visibility: public

Description: `Convert` will convert the bytes in `Bytes` from encoding `Source` to encoding `Destination` and returns the converted bytes as an array. If `StartIndex` and `Count` are specified, the conversion starts at (zero-based) index `StartIndex`, and only `Count` bytes will be converted.

See also: TEncoding.GetBytes (1869), TEncoding.GetChars (1869)

76.93.6 TEncoding.IsStandardEncoding

Synopsis: Check if the encoding is one of the standard encodings.

Declaration:
```
class function IsStandardEncoding(AEncoding: TEncoding) : Boolean
                                          ;  Static
```

Visibility: public

Description: `IsStandardEncoding` will return `True` is one of the standard encoding instances TEncoding.ANSI (1874), TEncoding.ASCII (1874), TEncoding.Default (1875), TEncoding.BigEndianUnicode (1874), TEncoding.Unicode (1875), TEncoding.UTF7 (1875) or TEncoding.UTF8 (1876). Otherwise it returns `False`.

See also: TEncoding.ANSI (1874), TEncoding.ASCII (1874), TEncoding.Default (1875), TEncoding.BigEndianUnicode (1874), TEncoding.Unicode (1875), TEncoding.UTF7 (1875), TEncoding.UTF8 (1876)

76.93.7 TEncoding.GetBufferEncoding

Synopsis: Attempt to guess the encoding of a buffer.

Declaration:
```
class function GetBufferEncoding(const Buffer: TBytes;
                                 var AEncoding: TEncoding) : Integer
                                 ;  Overload;  Static
class function GetBufferEncoding(const Buffer: TBytes;
                                 var AEncoding: TEncoding;
                                 ADefaultEncoding: TEncoding) : Integer
                                 ;  Overload;  Static
```

Visibility: public

Description: `GetBufferEncoding` checks the BOM marker of an array of bytes `Buffer`, and returns the found encoding in `AEncoding`. Only the TEncoding.BigEndianUnicode (1874)TEncoding.Unicode (1875) and TEncoding.UTF8 (1876) encodings will be tried.

If `AEncoding` is non-Nil on entry, it will be used to check the BOM marker with. No other encoding will be tried in that case.

If `ADefaultEncoding` is specified, it will be returned in case no match was found.

The function returns the number of bytes in the BOM Marker.

See also: TEncoding.BigEndianUnicode (1874), TEncoding.Unicode (1875), TEncoding.UTF8 (1876), TEncoding.GetPreamble (1872)

76.93.8 TEncoding.GetEncoding

Synopsis: Get an encoding instance for a given codepage.

Declaration:
```
class function GetEncoding(CodePage: Integer) : TEncoding;  Overload
                                ;  Static
class function GetEncoding(const EncodingName: UnicodeString)
                                : TEncoding;  Overload;  Static
```

Visibility: public

Description: `GetEncoding` will return an instance of `TEncoding` for the given codepage `CodePage`. The codepage can also be specified by name `EncodingName`. The returned instance is an appropriate descendent of `TEncoding` and needs to be freed by the caller.

See also: TUnicodeEncoding (1948), TBigEndianUnicodeEncoding (1851), TUTF8Encoding (1957), TUTF7Encoding (1956), TMBCSEncoding (1894)

76.93.9 TEncoding.GetMaxByteCount

Synopsis: Returns the maximum number of bytes needed to represent a string.

Declaration:
```
function GetMaxByteCount(CharCount: Integer) : Integer;  Virtual
                                ;  Abstract
```

Visibility: public

Description: `GetMaxByteCount` returns the maximum number of bytes needed to represent a string of `CharCount` characters in the given encoding.

This is an abstract method, implemented by descendents of `TEncoding`.

See also: TEncoding.GetMaxCharCount (1872)

76.93.10 TEncoding.GetMaxCharCount

Synopsis: Return the maximum number of characters that can be represented in a number of bytes.

Declaration:
```
function GetMaxCharCount(ByteCount: Integer) : Integer;  Virtual
                                ;  Abstract
```

Visibility: public

Description: `GetMaxCharCount` returns the theoretical maximum number of characters that can be represented in a buffer of length `ByteCount` in the given encoding. Note that the actual number of characters that can be represented may well be much less and depends on the encoding and the actual characters.

This is an abstract method, implemented by descendents of `TEncoding`.

See also: TEncoding.GetMaxByteCount (1872)

76.93.11 TEncoding.GetPreamble

Synopsis: Return the BOM Marker used by the encoding.

Declaration:
```
function GetPreamble : TBytes;  Virtual;  Abstract
```

Visibility: public

Description: `GetPreamble` returns the BOM marker bytes used by the encoding. This is only meaningful for the UTF8 and Unicode encodings, for all other encodings there is no BOM Marker.

See also: TUnicodeEncoding (1948), TBigEndianUnicodeEncoding (1851), TUTF8Encoding (1957)

76.93.12 TEncoding.GetString

Synopsis: Return a string based on an array of bytes.

Declaration:
```
function GetString(const Bytes: TBytes) : UnicodeString;  Overload
function GetString(const Bytes: TBytes; ByteIndex: Integer;
                   ByteCount: Integer) : UnicodeString;  Overload
```

Visibility: public

Description: `GetString` will return a Unicode string, created from the bytes in the `Bytes` array. The bytes array will be interpreted according to the encoding which the `TEncoding` represents.

If `ByteIndex` and `ByteCount` are specified, only the `ByteCount` bytes starting at position `ByteIndex` will be converted.

Errors: In case of invalid bytes, an `EEncodingError` exception may be raised.

76.93.13 TEncoding.CodePage

Synopsis: Codepage for this encoding.

Declaration: `Property CodePage : Cardinal`

Visibility: public

Access: Read

Description: `CodePage` is the numerical codepage for this encoding. It is a number as used in the Windows codepage registry.

See also: TEncoding.EncodingName (1873)

76.93.14 TEncoding.EncodingName

Synopsis: Name of this encoding.

Declaration: `Property EncodingName : UnicodeString`

Visibility: public

Access: Read

Description: `EncodingName` is the name for this encoding. It's based on the windows name for the encoding and is calculated from the codepage.

See also: TEncoding.CodePage (1873)

76.93.15 TEncoding.IsSingleByte

Synopsis: Is the encoding a single-byte encoding or not ?

Declaration: `Property IsSingleByte : Boolean`

Visibility: public

Access: Read

Description: `IsSingleByte` determines whether an encoding is single-byte or not. It is `false` for all standard encodings.

76.93.16 TEncoding.ANSI

Synopsis: Ansi encoding instance.

Declaration: `Property ANSI : TEncoding`

Visibility: public

Access: Read

Description: `ANSI` is the ANSI codepage encoding instance, it is the default single-byte string codepage on windows (as returned by DefaultSystemCodePage (1635)). This instance is created and maintained by the system, it should not be freed.

See also: (**??**), TEncoding.ASCII (1874), TEncoding.Default (1875), TEncoding.BigEndianUnicode (1874), TEncoding.Unicode (1875), TEncoding.UTF7 (1875), TEncoding.UTF8 (1876)

76.93.17 TEncoding.ASCII

Synopsis: ASCII encoding instance.

Declaration: `Property ASCII : TEncoding`

Visibility: public

Access: Read

Description: `ASCII` is the ASCII codepage (CP_ASCII) encoding instance. This instance is created and maintained by the system, it should not be freed.

See also: TEncoding.ANSI (1874), TEncoding.ASCII (1874), TEncoding.Default (1875), TEncoding.BigEndianUnicode (1874), TEncoding.Unicode (1875), TEncoding.UTF7 (1875), TEncoding.UTF8 (1876)

76.93.18 TEncoding.BigEndianUnicode

Synopsis: Big-endian Unicode (UTF16BE) encoding instance.

Declaration: `Property BigEndianUnicode : TEncoding`

Visibility: public

Access: Read

Description: `BigEndianUnicode` is the Big-endian Unicode encoding instance (CP_UTF16BE) This instance is created and maintained by the system, it should not be freed.

See also: (**??**), TEncoding.ANSI (1874), TEncoding.ASCII (1874), TEncoding.Default (1875), TEncoding.Unicode (1875), TEncoding.UTF7 (1875), TEncoding.UTF8 (1876)

76.93.19 TEncoding.Default

Synopsis: Default codepage.

Declaration: `Property Default : TEncoding`

Visibility: public

Access: Read

Description: `Defaulte` is the default encoding instance (it equals the ANSI codepage). This instance is created and maintained by the system, it should not be freed.

if the DefaultSystemCodePage (1443) changes, you should call TEncoding.FreeEncodings (1869) to regenerate the default encoding using the new code page.

See also: (**??**), TEncoding.ANSI (1874), TEncoding.ASCII (1874), TEncoding.BigEndianUnicode (1874), TEncoding.Unicode (1875), TEncoding.UTF7 (1875), TEncoding.UTF8 (1876)

76.93.20 TEncoding.SystemEncoding

Synopsis: System encoding.

Declaration: `Property SystemEncoding : TEncoding`

Visibility: public

Access: Read

Description: `SystemEncoding` is the encoding as referenced by the `DefaultSystemCodePage`.

See also: DefaultSystemCodePage (1443)

76.93.21 TEncoding.Unicode

Synopsis: UTF16 encoding instance.

Declaration: `Property Unicode : TEncoding`

Visibility: public

Access: Read

Description: `Unicode` is the Big-endian Unicode encoding instance (CP_UTF16) This instance is created and maintained by the system, it should not be freed.

See also: (**??**), TEncoding.ANSI (1874), TEncoding.ASCII (1874), TEncoding.Default (1875), TEncoding.BigEndianUnicode (1874), TEncoding.Unicode (1875), TEncoding.UTF7 (1875), TEncoding.UTF8 (1876)

76.93.22 TEncoding.UTF7

Synopsis: UTF7 encoding instance.

Declaration: `Property UTF7 : TEncoding`

Visibility: public

Access: Read

Description: `UTF7` is the UTF7 encoding instance (CP_UTF7) This instance is created and maintained by the system, it should not be freed.

See also: (**??**), TEncoding.ANSI (1874), TEncoding.ASCII (1874), TEncoding.Default (1875), TEncoding.BigEndianUnicode (1874), TEncoding.Unicode (1875), TEncoding.UTF8 (1876)

76.93.23 TEncoding.UTF8

Synopsis: UTF8 encoding instance.

Declaration: `Property UTF8 : TEncoding`

Visibility: public

Access: Read

Description: `UTF8` is the UTF7 encoding instance (CP_UTF7) This instance is created and maintained by the system, it should not be freed.

See also: (**??**), TEncoding.ANSI (1874), TEncoding.ASCII (1874), TEncoding.Default (1875), TEncoding.BigEndianUnicode (1874), TEncoding.Unicode (1875), TEncoding.UTF7 (1875)

76.94 TExtendedHelper

76.94.1 Description

`TExtendedHelper` is the helper type for the Extended-sized floating point type. It contains some conversion methods, as well as access to the low-level structure of the floating-point representation of a Extended.

See also: TDoubleHelper (1863), TExtendedHelper (1876)

76.94.2 Method overview

Page	Method	Description
1880	BuildUp	Build a Extended-sized floating point from its composing parts.
1880	Exponent	Exponent of the floating-point value.
1880	Fraction	Fraction of the floating-point value.
1878	IsInfinity	Check whether a value is positive or negative infinity.
1878	IsNan	Check whether a value equals `NaN`.
1879	IsNegativeInfinity	Check whether a value is negative infinity.
1879	IsPositiveInfinity	Check whether a value is positive infinity.
1880	Mantissa	Mantissa of the floating-point.
1877	Parse	Convert a string to a floating point value.
1879	Size	Size (in bytes) of a Extended-sized floating point value.
1881	SpecialType	Return the type of the Extended-sized floating point value.
1877	ToString	Convert a Extended-sized floating point value to a string.
1878	TryParse	Try to convert a string to a Extended-sized floating point value.

76.94.3 Property overview

Page	Properties	Access	Description
1881	Bytes	rw	Indexed access to the individual bytes of the floating point value.
1882	Exp	rw	The bit pattern of the exponent as stored in memory.
1882	Frac	rw	Bitpattern that makes up the fractional part.
1881	Sign	rw	Sign of the floating point value.
1881	Words	rw	Indexed access to the words that make up the floating point value.

76.94.4 TExtendedHelper.ToString

Synopsis: Convert a Extended-sized floating point value to a string.

Declaration:
```
class function ToString(const AValue: Extended) : string;  Overload
                        ;  Static
class function ToString(const AValue: Extended;
                        const AFormatSettings: TFormatSettings) : string
                        ;  Overload;  Static
class function ToString(const AValue: Extended;
                        const AFormat: TFloatFormat;
                        const APrecision: Integer;
                        const ADigits: Integer) : string;  Overload
                        ;  Static
class function ToString(const AValue: Extended;
                        const AFormat: TFloatFormat;
                        const APrecision: Integer;
                        const ADigits: Integer;
                        const AFormatSettings: TFormatSettings) : string
                        ;  Overload;  Static
function ToString(const AFormat: TFloatFormat;
                  const APrecision: Integer; const ADigits: Integer)
                   : string;  Overload
function ToString(const AFormat: TFloatFormat;
                  const APrecision: Integer; const ADigits: Integer;
                  const AFormatSettings: TFormatSettings) : string
                  ;  Overload
function ToString(const AFormatSettings: TFormatSettings) : string
                  ;  Overload
function ToString : string;  Overload
```

Visibility: public

Description: `ToString` will convert `AValue` (or `Self` in the plain method version) to a string. Optionally `FormatSettings` can be specified, to be able to specify the decimal separator character to use.

Additionally, a precision `APrecision` and number of digits `ADigits` can be specified, in conjunction with a `AFormat` parameter to specify the form in which the floating-point value must be represented. (see TFloatFormat (1656) for an explanation of the various values). In this case, FloatToStrF (1731) is used to format the value. In the absence of these parameters, FloatToStr (1729) is called.

See also: FloatToStr (1729), FloatToStrF (1731), TFloatFormat (1656)

76.94.5 TExtendedHelper.Parse

Synopsis: Convert a string to a floating point value.

Declaration:
```
class function Parse(const AString: string) : Extended;  Overload
                     ;  Static
class function Parse(const AString: string;
                     const AFormatSettings: TFormatSettings) : Extended
                     ;  Overload;  Static
```

Visibility: public

Description: `Parse` will try to convert `AString` to a Extended-sized floating point value. It will take into account internationalization settings. (it uses `FloatToStr`).

Errors: If the string `AString` is not a valid floating-point value, a EConvertError (1830) exception is raised.

See also: FloatToStr (1729), TExtendedHelper.ToString (1877), TExtendedHelper.TryParse (1878)

76.94.6 TExtendedHelper.TryParse

Synopsis: Try to convert a string to a Extended-sized floating point value.

Declaration:
```
class function TryParse(const AString: string; out AValue: Extended)
                        : Boolean;  Overload;  Static
class function TryParse(const AString: string; out AValue: Extended;
                        const AFormatSettings: TFormatSettings) : Boolean
                        ;  Overload;  Static
```

Visibility: public

Description: `TryParse` attempts to convert the string `AString` to a Extended-sized floating point value and reports `True` if the conversion was successful. In that case the parsed value is returned in `AValue`.

If the conversion failed, `False` is returned.

See also: TExtendedHelper.Parse (1877), TExtendedHelper.ToString (1877)

76.94.7 TExtendedHelper.IsNan

Synopsis: Check whether a value equals `NaN`.

Declaration:
```
class function IsNan(const AValue: Extended) : Boolean;  Overload
                     ;  Static
function IsNan : Boolean;  Overload
```

Visibility: public

Description: `IsNan` checks whether a Extended-sized floating point value is `NaN` (Not a Number). If so, it returns `True`. When the class function version is used, the value can be specified using `AValue`. In the method version, the used value is (`Self`).

See also: TExtendedHelper.IsInfinity (1878), TExtendedHelper.IsPositiveInfinity (1879), TExtendedHelper.IsNegativeInfinity (1879)

76.94.8 TExtendedHelper.IsInfinity

Synopsis: Check whether a value is positive or negative infinity.

Declaration:
```
class function IsInfinity(const AValue: Extended) : Boolean;  Overload
                          ;  Static
function IsInfinity : Boolean;  Overload
```

Visibility: public

Description: `IsInfinity` checks whether a Extended-sized floating point value represents a positive or negative infinity. If so, it returns `True`. When the class function version is used, the value can be specified using `AValue`. In the method version, the used value is (`Self`).

See also: TExtendedHelper.IsNan (1878), TExtendedHelper.IsPositiveInfinity (1879), TExtendedHelper.IsNegativeInfinity (1879)

76.94.9 TExtendedHelper.IsNegativeInfinity

Synopsis: Check whether a value is negative infinity.

Declaration:
```
class function IsNegativeInfinity(const AValue: Extended) : Boolean
                                  ;  Overload;  Static
function IsNegativeInfinity : Boolean;  Overload
```

Visibility: public

Description: `IsNegativeInfinity` checks whether a Extended-sized floating point value represents a negative infinity. If so, it returns `True`. When the class function version is used, the value can be specified using `AValue`. In the method version, the used value is (`Self`).

See also: TExtendedHelper.IsNan (1878), TExtendedHelper.IsPositiveInfinity (1879), TExtendedHelper.IsInfinity (1878)

76.94.10 TExtendedHelper.IsPositiveInfinity

Synopsis: Check whether a value is positive infinity.

Declaration:
```
class function IsPositiveInfinity(const AValue: Extended) : Boolean
                                  ;  Overload;  Static
function IsPositiveInfinity : Boolean;  Overload
```

Visibility: public

Description: `IsPositiveInfinity` checks whether a Extended-sized floating point value represents a positive infinity. If so, it returns `True`. When the class function version is used, the value can be specified using `AValue`. In the method version, the used value is (`Self`).

See also: TExtendedHelper.IsNan (1878), TExtendedHelper.IsNegativeInfinity (1879), TExtendedHelper.IsInfinity (1878)

76.94.11 TExtendedHelper.Size

Synopsis: Size (in bytes) of a Extended-sized floating point value.

Declaration:
```
class function Size : Integer;  Static
```

Visibility: public

Description: `Size` is the size (in bytes) of a Extended-sized floating point value. It is equivalent to calling `SizeOf(Extended)`.

See also: SizeOf (1574)

76.94.12 TExtendedHelper.BuildUp

Synopsis: Build a Extended-sized floating point from its composing parts.

Declaration:
```
procedure BuildUp(const ASignFlag: Boolean; const AMantissa: QWord;
                  const AExponent: Integer)
```

Visibility: public

Description: `BuildUp` will compose a Extended-sized floating point value from the sign `ASignFlag`, mantissa `AMantissa` and exponent `AExponent`. It simply sets the Sign (1881), Exp (1882) and Frac (1882) properties in 1 call.

See also: TExtendedHelper.Sign (1881), TExtendedHelper.Exp (1882), TExtendedHelper.Frac (1882)

76.94.13 TExtendedHelper.Exponent

Synopsis: Exponent of the floating-point value.

Declaration:
```
function Exponent : Integer
```

Visibility: public

Description: `Exponent` is the value `X` in the representation of the floating-point value in $m*2\hat{X}$, i.e. the exponent.

See also: TExtendedHelper.Sign (1881), TExtendedHelper.Exp (1882), TExtendedHelper.Frac (1882), TExtendedHelper.Fraction (1880), TExtendedHelper.Mantissa (1880)

76.94.14 TExtendedHelper.Fraction

Synopsis: Fraction of the floating-point value.

Declaration:
```
function Fraction : Extended
```

Visibility: public

Description: `Fraction` is the decimal part of the floating-point value.

See also: TExtendedHelper.Sign (1881), TExtendedHelper.Exp (1882), TExtendedHelper.Exponent (1880), TExtendedHelper.Frac (1882), TExtendedHelper.Mantissa (1880)

76.94.15 TExtendedHelper.Mantissa

Synopsis: Mantissa of the floating-point.

Declaration:
```
function Mantissa : QWord
```

Visibility: public

Description: `Mantissa` is the value of the significand without the hidden bit. This means it the plain bit pattern as it is stored in memory.

See also: TExtendedHelper.Sign (1881), TExtendedHelper.Exp (1882), TExtendedHelper.Exponent (1880), TExtendedHelper.Frac (1882), TExtendedHelper.Fraction (1880)

76.94.16 TExtendedHelper.SpecialType

Synopsis: Return the type of the Extended-sized floating point value.

Declaration: `function SpecialType : TFloatSpecial`

Visibility: public

Description: `SpecialType` checks whether the Extended-sized floating point value equals one of several special values, and returns an enumerated value describing which value this is. See TFloatSpecial (1635) for an explanation of the possible values.

See also: TFloatSpecial (1635)

76.94.17 TExtendedHelper.Bytes

Synopsis: Indexed access to the individual bytes of the floating point value.

Declaration: `Property Bytes[AIndex: Cardinal]: Byte`

Visibility: public

Access: Read,Write

Description: `Bytes` can be used to get or set the various bytes that make up the Extended-sized floating point value. The index runs from 0 to `Size-1`.

See also: TExtendedHelper.Words (1881), TExtendedHelper.Size (1879)

76.94.18 TExtendedHelper.Words

Synopsis: Indexed access to the words that make up the floating point value.

Declaration: `Property Words[AIndex: Cardinal]: Word`

Visibility: public

Access: Read,Write

Description: `Words` can be used to get or set the various bytes that make up the Extended-sized floating point value. The index runs from 0 to `(Size-1) div 2`.

See also: TExtendedHelper.Bytes (1881), TExtendedHelper.Size (1879)

76.94.19 TExtendedHelper.Sign

Synopsis: Sign of the floating point value.

Declaration: `Property Sign : Boolean`

Visibility: public

Access: Read,Write

Description: `Sign` returns `True` if the sign bit of the value is set (i.e. it is a negative value) or `False` if it is not set (i.e. it is a positive value).

See also: TExtendedHelper.Bytes (1881), TExtendedHelper.Exp (1882), TExtendedHelper.Frac (1882), TExtendedHelper.Mantissa (1880), TExtendedHelper.Fraction (1880), TExtendedHelper.Exponent (1880)

76.94.20 TExtendedHelper.Exp

Synopsis: The bit pattern of the exponent as stored in memory.

Declaration:
```
Property Exp : QWord
```

Visibility: public

Access: Read,Write

Description: `Exp` is the internal representation of the Exponent (1880).

See also: TExtendedHelper.Bytes (1881), TExtendedHelper.Sign (1881), TExtendedHelper.Frac (1882), TExtendedHelper.Mantissa (1880), TExtendedHelper.Fraction (1880), TExtendedHelper.Exponent (1880)

76.94.21 TExtendedHelper.Frac

Synopsis: Bitpattern that makes up the fractional part.

Declaration:
```
Property Frac : QWord
```

Visibility: public

Access: Read,Write

Description: `Frac` is the bit pattern representing the fractional part (significand) including the preceding 1 (the hidden bit).

See also: TExtendedHelper.Bytes (1881), TExtendedHelper.Sign (1881), TExtendedHelper.Exp (1882), TExtendedHelper.Mantissa (1880), TExtendedHelper.Fraction (1880), TExtendedHelper.Exponent (1880)

76.95 TGuidHelper

76.95.1 Description

`TGuidHelper` provides various methods for a TGUID (1635) type.

See also: TGUID (1635)

76.95.2 Method overview

Page	Method	Description
1882	Create	Initialize a `TGUID` instance from data.
1883	NewGuid	Create a new GUID.
1883	ToByteArray	Convert to array of bytes.
1884	ToString	Convert to string.

76.95.3 TGuidHelper.Create

Synopsis: Initialize a `TGUID` instance from data.

Declaration:
```
class function Create(const Data; BigEndian: Boolean) : TGuid;  Overload
                      ;  Static
class function Create(const Data: Array of Byte; AStartIndex: Cardinal;
                      BigEndian: Boolean) : TGuid;  Overload;  Static
class function Create(const Data; DataEndian: TEndian) : TGuid
```

```
                    ;  Overload;  Static
class function Create(const B: TBytes; DataEndian: TEndian) : TGuid
                    ;  Overload;  Static
class function Create(const B: TBytes; AStartIndex: Cardinal;
                    DataEndian: TEndian) : TGuid;  Overload;  Static
class function Create(const S: string) : TGuid;  Overload;  Static
class function Create(A: Integer; B: SmallInt; C: SmallInt;
                    const D: TBytes) : TGuid;  Overload;  Static
class function Create(A: Integer; B: SmallInt; C: SmallInt; D: Byte;
                    E: Byte; F: Byte; G: Byte; H: Byte; I: Byte;
                    J: Byte; K: Byte) : TGuid;  Overload;  Static
class function Create(A: Cardinal; B: Word; C: Word; D: Byte; E: Byte;
                    F: Byte; G: Byte; H: Byte; I: Byte; J: Byte;
                    K: Byte) : TGuid;  Overload;  Static
```

Visibility: default

Description: `Create` will initialize a TGUID (1635) from byte data. The byte-data can come in different forms

Datacan be a plain buffer. The GUID Data will be read from the buffer (`SizeOf(TGUID)` bytes), according to the `BigEndian` argument.

Datacan be an array of bytes. `SizeOf(TGUID)` bytes will be read from the array, starting at index `StartIndex` (default 0). The GUID Data will be read according to the `BigEndian` argument.

Scan be a string containing a string representation of the GUID, which is converted using StringToGUID (1779).

A,B,C,D (optionally E,F,G,H,I,J,K)where the arguments are simply the various components of the GUID.

See also: TGUID (1635), StringToGUID (1779)

76.95.4 TGuidHelper.NewGuid

Synopsis: Create a new GUID.

Declaration: `class function NewGuid : TGuid;  Static`

Visibility: default

Description: `NewGUID` creates and returns a new (**??**) using CreateGUID (1693)

See also: CreateGUID (1693)

76.95.5 TGuidHelper.ToByteArray

Synopsis: Convert to array of bytes.

Declaration: `function ToByteArray(DataEndian: TEndian) : TBytes`

Visibility: default

Description: `ToByteArray` converts the TGUID to an array of bytes (of length `SizeOf(TGUID)`). The bytes will be ordered according to `DataEndian`.

See also: TGUIDHelper.ToString (1884), TGUIDHelper.Create (1882)

76.95.6 TGuidHelper.ToString

Synopsis: Convert to string.

Declaration: `function ToString(SkipBrackets: Boolean) : string`

Visibility: default

Description: `ToString` will convert the TGUID to a string representation, using GUIDToString (1754)

See also: GUIDToString (1754)

76.96 TInt64Helper

76.96.1 Description

`TInt64Helper` contains some auxiliary routines for a Int64-typed ordinal value. It consists mainly of conversion routines to and from other types.

See also: TStringHelper (1926), TShortIntHelper (1910), TSmallIntHelper (1922), TWordHelper (1960), TCardinalHelper (1859), TIntegerHelper (1888), TByteHelper (1855), TQWordHelper (1906), TNativeIntHelper (1898), TNativeUIntHelper (1902)

76.96.2 Method overview

Page	Method	Description
1887	ClearBit	Set bit to 0.
1884	Parse	Convert from a string.
1887	SetBit	Set bit to 1.
1885	Size	Size, in bytes, of the Int64 value.
1888	TestBit	Check bit.
1886	ToBinString	Convert to a binary string representation.
1885	ToBoolean	Convert to a boolean value.
1886	ToDouble	Convert to a double-sized floating point value.
1886	ToExtended	Convert to an extended-sized floating point value.
1887	ToggleBit	Invert bit.
1886	ToHexString	Convert to a hexadecimal string representation.
1887	ToSingle	Convert to an single-sized floating point value.
1885	ToString	Convert the value to string.
1885	TryParse	Try to convert a string to a Int64, report success or failure.

76.96.3 TInt64Helper.Parse

Synopsis: Convert from a string.

Declaration: `class function Parse(const AString: string) : Int64;  Static`

Visibility: public

Description: `Parse` will attempt to convert the string `AString` to a Int64 value. It uses the StrToInt64 (1794) function to perform the conversion, so no localization is taken into account.

Errors: If the string does not contain a valid Int64 value, an EConvertError (1830) exception is raised.

See also: TInt64Helper.ToString (1885), TInt64Helper.TryParse (1885), StrToInt64 (1794)

76.96.4 TInt64Helper.Size

Synopsis: Size, in bytes, of the Int64 value.

Declaration:
```
class function Size : Integer;  Static
```

Visibility: public

Description: `Size` returns the size (in Int64s) of the Int64 value. This is equivalent to `SizeOf(Int64)`.

Errors: None.

See also: SizeOf (1574)

76.96.5 TInt64Helper.ToString

Synopsis: Convert the value to string.

Declaration:
```
class function ToString(const AValue: Int64) : string;  Overload
                              ;  Static
function ToString : string;  Overload
```

Visibility: public

Description: `ToString` will, in the class function variant of this method, convert `AValue` to a string representation. In the regular method overloaded version of `ToString`, the Int64 value itself is used. The IntToStr (1758) function is used to do the conversion.

See also: TInt64Helper.Parse (1884), IntToStr (1758)

76.96.6 TInt64Helper.TryParse

Synopsis: Try to convert a string to a Int64, report success or failure.

Declaration:
```
class function TryParse(const AString: string; out AValue: Int64)
                               : Boolean;  Static
```

Visibility: public

Description: `TryParse` attempts to convert the string `AString` to a Int64, and reports the success of the attempt. If the attempt is successful, then `True` is returned, and the actual value of the Int64 is returned in `AValue`.

It uses the val (1635) function to perform the conversion, so no localization is taken into account.

See also: TInt64Helper.Parse (1884), Val (1598)

76.96.7 TInt64Helper.ToBoolean

Synopsis: Convert to a boolean value.

Declaration:
```
function ToBoolean : Boolean
```

Visibility: public

Description: `ToBoolean` converts the Int64 value to a boolean: it returns `True` if the value is nonzero, `False` if it is zero.

See also: TInt64Helper.ToSingle (1887), TInt64Helper.ToDouble (1886), TInt64Helper.ToExtended (1886), TInt64Helper.ToString (1885), TInt64Helper.ToHexString (1886)

76.96.8 TInt64Helper.ToDouble

Synopsis: Convert to a double-sized floating point value.

Declaration: `function ToDouble : Double`

Visibility: public

Description: `ToDouble` converts the Int64 value to a double-sized floating point value.

See also: TInt64Helper.ToBoolean (1885), TInt64Helper.ToExtended (1886), TInt64Helper.ToSingle (1887), TInt64Helper.ToString (1885), TInt64Helper.ToHexString (1886)

76.96.9 TInt64Helper.ToExtended

Synopsis: Convert to an extended-sized floating point value.

Declaration: `function ToExtended : Extended`

Visibility: public

Description: `ToDouble` converts the Int64 value to an extended-sized floating point value.

See also: TInt64Helper.ToBoolean (1885), TInt64Helper.ToSingle (1887), TInt64Helper.ToDouble (1886), TInt64Helper.ToString (1885), TInt64Helper.ToHexString (1886)

76.96.10 TInt64Helper.ToBinString

Synopsis: Convert to a binary string representation.

Declaration: `function ToBinString : string`

Visibility: public

Description: `ToBinString` converts the Int64 value to a binary string representation, showing all 64 bits.

See also: TInt64Helper.ToHexString (1886), TInt64Helper.ToString (1885)

76.96.11 TInt64Helper.ToHexString

Synopsis: Convert to a hexadecimal string representation.

Declaration:
```
function ToHexString(const AMinDigits: Integer) : string;  Overload
function ToHexString : string;  Overload
```

Visibility: public

Description: `ToHexString` converts the Int64 value to a hexadecimal string representation. The `AMinDigits` argument specifies the minimal number of characters in the resulting string. The string will be left-padded with zeroes if the representation contains less than `AMinDigits` characters.

See also: TInt64Helper.ToBoolean (1885), TInt64Helper.ToSingle (1887), TInt64Helper.ToDouble (1886), TInt64Helper.ToString (1885), TInt64Helper.ToExtended (1886)

76.96.12 TInt64Helper.ToSingle

Synopsis: Convert to an single-sized floating point value.

Declaration: `function ToSingle : Single`

Visibility: public

Description: `ToSingle` converts the Int64 value to a single-sized floating point value.

See also: TInt64Helper.ToBoolean (1885), TInt64Helper.ToDouble (1886), TInt64Helper.ToExtended (1886), TInt64Helper.ToString (1885), TInt64Helper.ToHexString (1886)

76.96.13 TInt64Helper.SetBit

Synopsis: Set bit to 1.

Declaration: `function SetBit(const Index: TInt64BitIndex) : Int64`

Visibility: public

Description: `SetBit` puts value 1 to bit number determined by argument `Index`.

See also: TInt64Helper.ClearBit (1887), TInt64Helper.ToggleBit (1887), TInt64Helper.TestBit (1888), TInt64Helper.HighestSetBitPos (1884), TInt64Helper.LowestSetBitPos (1884), TInt64Helper.SetBitsCount (1884), TInt64Helper.Bits (1884)

76.96.14 TInt64Helper.ClearBit

Synopsis: Set bit to 0.

Declaration: `function ClearBit(const Index: TInt64BitIndex) : Int64`

Visibility: public

Description: `ClearBit` puts value 0 to bit number determined by argument `Index`.

See also: TInt64Helper.SetBit (1887), TInt64Helper.ToggleBit (1887), TInt64Helper.TestBit (1888), TInt64Helper.HighestSetBitPos (1884), TInt64Helper.LowestSetBitPos (1884), TInt64Helper.SetBitsCount (1884), TInt64Helper.Bits (1884)

76.96.15 TInt64Helper.ToggleBit

Synopsis: Invert bit.

Declaration: `function ToggleBit(const Index: TInt64BitIndex) : Int64`

Visibility: public

Description: `ToggleBit` reads bit value from bit number determined by argument `Index`, inverts it (if it was 0 it becomes 1, and vice versa), and stores it back.

See also: TInt64Helper.SetBit (1887), TInt64Helper.ToggleBit (1887), TInt64Helper.TestBit (1888), TInt64Helper.HighestSetBitPos (1884), TInt64Helper.LowestSetBitPos (1884), TInt64Helper.SetBitsCount (1884), TInt64Helper.Bits (1884)

76.96.16 TInt64Helper.TestBit

Synopsis: Check bit.

Declaration: `function TestBit(const Index: TInt64BitIndex) : Boolean`

Visibility: public

Description: `TestBit` reads boolean value (true if bit is 1, and false if bit is 0) from bit number determined by argument `Index`.

See also: TInt64Helper.SetBit (1887), TInt64Helper.ToggleBit (1887), TInt64Helper.TestBit (1888), TInt64Helper.HighestSetBitPos (1884), TInt64Helper.LowestSetBitPos (1884), TInt64Helper.SetBitsCount (1884), TInt64Helper.Bits (1884)

76.97 TIntegerHelper

76.97.1 Description

`TIntegerHelper` contains some auxiliary routines for a Integer-typed ordinal value. It consists mainly of conversion routines to and from other types.

See also: TStringHelper (1926), TShortIntHelper (1910), TSmallIntHelper (1922), TWordHelper (1960), TCardinalHelper (1859), TByteHelper (1855), TInt64Helper (1884), TQWordHelper (1906), TNativeIntHelper (1898), TNativeUIntHelper (1902)

76.97.2 Method overview

Page	Method	Description
1891	ClearBit	Set bit to 0.
1889	Parse	Convert from a string.
1891	SetBit	Set bit to 1.
1888	Size	Size, in bytes, of the Integer value.
1892	TestBit	Check bit.
1890	ToBinString	Convert to a binary string representation.
1889	ToBoolean	Convert to a boolean value.
1890	ToDouble	Convert to a double-sized floating point value.
1890	ToExtended	Convert to an extended-sized floating point value.
1891	ToggleBit	Invert bit.
1890	ToHexString	Convert to a hexadecimal string representation.
1891	ToSingle	Convert to an single-sized floating point value.
1889	ToString	Convert the value to string.
1889	TryParse	Try to convert a string to a Integer, report success or failure.

76.97.3 TIntegerHelper.Size

Synopsis: Size, in bytes, of the Integer value.

Declaration: `class function Size : Integer;  Static`

Visibility: public

Description: `Size` returns the size (in Integers) of the Integer value. This is equivalent to `SizeOf(Integer)`.

Errors: None.

See also: SizeOf (1574)

76.97.4 TIntegerHelper.ToString

Synopsis: Convert the value to string.

Declaration:
```
class function ToString(const AValue: Integer) : string;  Overload
                              ;  Static
function ToString : string;  Overload
```

Visibility: public

Description: `ToString` will, in the class function variant of this method, convert `AValue` to a string representation. In the regular method overloaded version of `ToString`, the Integer value itself is used. The IntToStr (1758) function is used to do the conversion.

See also: TIntegerHelper.Parse (1889), IntToStr (1758)

76.97.5 TIntegerHelper.Parse

Synopsis: Convert from a string.

Declaration:
```
class function Parse(const AString: string) : Integer;  Static
```

Visibility: public

Description: `Parse` will attempt to convert the string `AString` to a Integer value. It uses the StrToInt (1793) function to perform the conversion, so no localization is taken into account.

Errors: If the string does not contain a valid Integer value, an EConvertError (1830) exception is raised.

See also: TIntegerHelper.ToString (1889), TIntegerHelper.TryParse (1889), StrToInt (1793)

76.97.6 TIntegerHelper.TryParse

Synopsis: Try to convert a string to a Integer, report success or failure.

Declaration:
```
class function TryParse(const AString: string; out AValue: Integer)
                              : Boolean;  Static
```

Visibility: public

Description: `TryParse` attempts to convert the string `AString` to a Integer, and reports the success of the attempt. If the attempt is successful, then `True` is returned, and the actual value of the Integer is returned in `AValue`.

It uses the val (1635) function to perform the conversion, so no localization is taken into account.

See also: TIntegerHelper.Parse (1889), Val (1598)

76.97.7 TIntegerHelper.ToBoolean

Synopsis: Convert to a boolean value.

Declaration:
```
function ToBoolean : Boolean
```

Visibility: public

Description: `ToBoolean` converts the Integer value to a boolean: it returns `True` if the value is nonzero, `False` if it is zero.

See also: TIntegerHelper.ToSingle (1891), TIntegerHelper.ToDouble (1890), TIntegerHelper.ToExtended (1890), TIntegerHelper.ToString (1889), TIntegerHelper.ToHexString (1890)

76.97.8 TIntegerHelper.ToDouble

Synopsis: Convert to a double-sized floating point value.

Declaration: `function ToDouble : Double`

Visibility: public

Description: `ToDouble` converts the Integer value to a double-sized floating point value.

See also: TIntegerHelper.ToBoolean (1889), TIntegerHelper.ToExtended (1890), TIntegerHelper.ToSingle (1891), TIntegerHelper.ToString (1889), TIntegerHelper.ToHexString (1890)

76.97.9 TIntegerHelper.ToExtended

Synopsis: Convert to an extended-sized floating point value.

Declaration: `function ToExtended : Extended`

Visibility: public

Description: `ToDouble` converts the Integer value to an extended-sized floating point value.

See also: TIntegerHelper.ToBoolean (1889), TIntegerHelper.ToSingle (1891), TIntegerHelper.ToDouble (1890), TIntegerHelper.ToString (1889), TIntegerHelper.ToHexString (1890)

76.97.10 TIntegerHelper.ToBinString

Synopsis: Convert to a binary string representation.

Declaration: `function ToBinString : string`

Visibility: public

Description: `ToBinString` converts the Integer value to a binary string representation, showing all 32 bits.

See also: TIntegerHelper.ToHexString (1890), TIntegerHelper.ToString (1889)

76.97.11 TIntegerHelper.ToHexString

Synopsis: Convert to a hexadecimal string representation.

Declaration:
```
function ToHexString(const AMinDigits: Integer) : string;  Overload
function ToHexString : string;  Overload
```

Visibility: public

Description: `ToHexString` converts the Integer value to a hexadecimal string representation. The `AMinDigits` argument specifies the minimal number of characters in the resulting string. The string will be left-padded with zeroes if the representation contains less than `AMinDigits` characters.

See also: TIntegerHelper.ToBoolean (1889), TIntegerHelper.ToSingle (1891), TIntegerHelper.ToDouble (1890), TIntegerHelper.ToString (1889), TIntegerHelper.ToExtended (1890)

76.97.12 TIntegerHelper.ToSingle

Synopsis: Convert to an single-sized floating point value.

Declaration: `function ToSingle : Single`

Visibility: public

Description: `ToSingle` converts the Integer value to a single-sized floating point value.

See also: TIntegerHelper.ToBoolean (1889), TIntegerHelper.ToDouble (1890), TIntegerHelper.ToExtended (1890), TIntegerHelper.ToString (1889), TIntegerHelper.ToHexString (1890)

76.97.13 TIntegerHelper.SetBit

Synopsis: Set bit to 1.

Declaration: `function SetBit(const Index: TIntegerBitIndex) : Integer`

Visibility: public

Description: `SetBit` puts value 1 to bit number determined by argument `Index`.

See also: TIntegerHelper.ClearBit (1891), TIntegerHelper.ToggleBit (1891), TIntegerHelper.TestBit (1892), TIntegerHelper.HighestSetBitPos (1888), TIntegerHelper.LowestSetBitPos (1888), TIntegerHelper.SetBitsCount (1888), TIntegerHelper.Bits (1888)

76.97.14 TIntegerHelper.ClearBit

Synopsis: Set bit to 0.

Declaration: `function ClearBit(const Index: TIntegerBitIndex) : Integer`

Visibility: public

Description: `ClearBit` puts value 0 to bit number determined by argument `Index`.

See also: TIntegerHelper.SetBit (1891), TIntegerHelper.ToggleBit (1891), TIntegerHelper.TestBit (1892), TIntegerHelper.HighestSetBitPos (1888), TIntegerHelper.LowestSetBitPos (1888), TIntegerHelper.SetBitsCount (1888), TIntegerHelper.Bits (1888)

76.97.15 TIntegerHelper.ToggleBit

Synopsis: Invert bit.

Declaration: `function ToggleBit(const Index: TIntegerBitIndex) : Integer`

Visibility: public

Description: `ToggleBit` reads bit value from bit number determined by argument `Index`, inverts it (if it was 0 it becomes 1, and vice versa), and stores it back.

See also: TIntegerHelper.SetBit (1891), TIntegerHelper.ToggleBit (1891), TIntegerHelper.TestBit (1892), TIntegerHelper.HighestSetBitPos (1888), TIntegerHelper.LowestSetBitPos (1888), TIntegerHelper.SetBitsCount (1888), TIntegerHelper.Bits (1888)

76.97.16 TIntegerHelper.TestBit

Synopsis: Check bit.

Declaration: `function TestBit(const Index: TIntegerBitIndex) : Boolean`

Visibility: public

Description: `TestBit` reads boolean value (true if bit is 1, and false if bit is 0) from bit number determined by argument `Index`.

See also: TIntegerHelper.SetBit (1891), TIntegerHelper.ToggleBit (1891), TIntegerHelper.TestBit (1892), TIntegerHelper.HighestSetBitPos (1888), TIntegerHelper.LowestSetBitPos (1888), TIntegerHelper.SetBitsCount (1888), TIntegerHelper.Bits (1888)

76.98 TLongBoolHelper

76.98.1 Description

`TLongBoolHelper` is a helper type for the `LongBool` type. It contains mostly conversion routines to and from other types.

See also: TStringHelper (1926), TShortIntHelper (1910), TSmallIntHelper (1922), TWordHelper (1960), TCardinalHelper (1859), TIntegerHelper (1888), TInt64Helper (1884), TQWordHelper (1906), TNativeIntHelper (1898), TByteHelper (1855), TByteBoolHelper (1853), TWordBoolHelper (1959), TLongBoolHelper (1892)

76.98.2 Method overview

Page	Method	Description
1892	Parse	Convert string value to LongBool value.
1893	Size	Return the size (in bytes) of the.
1893	ToInteger	Convert to an integer value.
1893	ToString	Convert a LongBool value to string.
1893	TryToParse	Try to convert a string to a LongBool value.

76.98.3 TLongBoolHelper.Parse

Synopsis: Convert string value to LongBool value.

Declaration: `class function Parse(const S: string) : Boolean;  Static`

Visibility: public

Description: `Parse` attempts to convert the string `S` to a LongBool value. It uses the StrToBool (1787) function to perform the conversion.

Errors: If `S` does not contain a valid string representation, then an EConvertError (1830) exception is raised.

See also: TLongBoolHelper.TryToParse (1893), TLongBoolHelper.ToString (1893), TLongBoolHelper.ToInteger (1893)

76.98.4 TLongBoolHelper.Size

Synopsis: Return the size (in bytes) of the.

Declaration:
```
class function Size : Integer;  Static
```

Visibility: public

Description: `Size` returns the size (in bytes) of the LongBool value. This is equivalent to `SizeOf(LongBool)`.

See also: SizeOf (1574)

76.98.5 TLongBoolHelper.ToString

Synopsis: Convert a LongBool value to string.

Declaration:
```
class function ToString(const AValue: Boolean;
                        UseBoolStrs: TUseBoolStrs) : string;  Overload
                        ;  Static
function ToString(UseBoolStrs: TUseBoolStrs) : string;  Overload
```

Visibility: public

Description: `ToString` will, in the class method version, convert the `AValue` LongBool to a string representation. In the function method version the LongBool value itself (`Self`) will be converted.

If the `UseBoolStrs` parameter equals `TUseBoolStrs.True`, then the string representation will use the LongBool strings BoolStrs (1635). The default value for `UseBoolStrs` is `TUseBoolStrs.False`.

The conversion is done using the BoolTostr (1685) function.

See also: BoolStrs (1635), BoolToStr (1685)

76.98.6 TLongBoolHelper.TryToParse

Synopsis: Try to convert a string to a LongBool value.

Declaration:
```
class function TryToParse(const S: string; out AValue: Boolean)
                          : Boolean;  Static
```

Visibility: public

Description: `TryToParse` will attempt to convert the string `S` to a LongBool value. If the attempt is successful, `True` is returned, and the actual value is returned in `AValue`. If the attempt failed, `False` is returned.

See also: TLongBoolHelper.Parse (1892), TLongBoolHelper.ToString (1893)

76.98.7 TLongBoolHelper.ToInteger

Synopsis: Convert to an integer value.

Declaration:
```
function ToInteger : Integer
```

Visibility: public

Description: `ToInteger` will return the LongBool value, typecasted to `Integer`.

See also: TLongBoolHelper.ToString (1893)

76.99 TMBCSEncoding

76.99.1 Description

`TMBCSEncoding` is the encoding class used for most of the encodings, except the actual Unicode encodings. The codepage is specified during creation, and is by default the system codepage.

The name is somewhat misleading, since the MBCS is also used for single-byte encodings.

See also: TUnicodeEncoding (1948), TUTF8Encoding (1957), TUTF7Encoding (1956), TBigendianUnicodeEncoding (1851)

76.99.2 Method overview

Page	Method	Description
1894	Clone	Clone a `TMBCSEncoding` instance.
1894	Create	Create a new instance of a multi-byte character set encoding.
1894	GetMaxByteCount	Return max number of bytes needed to represent a string.
1895	GetMaxCharCount	Return max number of characters that can be represented by an array of bytes.
1895	GetPreamble	Return BOM marker bytes.

76.99.3 TMBCSEncoding.Create

Synopsis: Create a new instance of a multi-byte character set encoding.

Declaration:
```
constructor Create;  Virtual;  Overload
constructor Create(ACodePage: Integer);  Virtual;  Overload
constructor Create(ACodePage: Integer; MBToWCharFlags: Integer;
                   WCharToMBFlags: Integer);  Virtual;  Overload
```

Visibility: public

Description: `Create` instantiates a new instance of the multi-byte character set encoding. The `ACodePage` parameter is optional, and defaults to DefaultSystemCodePage (1635).

The `MBToWCharFlags` and `WCharToMBFlags` parameters are stored but are otherwise unused in the Free Pascal implementation of `TMBCSEncoding`

76.99.4 TMBCSEncoding.Clone

Synopsis: Clone a `TMBCSEncoding` instance.

Declaration:
```
function Clone : TEncoding;  Override
```

Visibility: public

Description: `Clone` overrides TEncoding.Clone (1870) to provide a clone of the `TMBCSEncoding` instance.

See also: TEncoding.Clone (1870)

76.99.5 TMBCSEncoding.GetMaxByteCount

Synopsis: Return max number of bytes needed to represent a string.

Declaration:
```
function GetMaxByteCount(CharCount: Integer) : Integer;  Override
```

Visibility: public

Description: `GetMaxByteCount` overrides TEncoding.GetMaxByteCount (1872) to return the maximum number of bytes needed to represent a string.

See also: TEncoding.GetMaxByteCount (1872)

76.99.6 TMBCSEncoding.GetMaxCharCount

Synopsis: Return max number of characters that can be represented by an array of bytes.

Declaration: `function GetMaxCharCount(ByteCount: Integer) : Integer;  Override`

Visibility: public

Description: `GetMaxCharCount` overrides TEncoding.GetMaxCharCount (1872) to return the maximum number of bytes needed to represent a string.

See also: TEncoding.GetMaxCharCount (1872)

76.99.7 TMBCSEncoding.GetPreamble

Synopsis: Return BOM marker bytes.

Declaration: `function GetPreamble : TBytes;  Override`

Visibility: public

Description: `GetPreamble` overrides TEncoding.GetPreamble (1872) to return the BOM Marker bytes (none, for this implementation).

See also: TEncoding.GetPreamble (1872)

76.100 TMREWException

76.100.1 Description

`TMREWException` is the exception used by the TMultiReadExclusiveWriteSynchronizer (1895) class, for example in case of a deadlock.

See also: TMultiReadExclusiveWriteSynchronizer (1895)

76.101 TMultiReadExclusiveWriteSynchronizer

76.101.1 Description

`TMultiReadExclusiveWriteSynchronizer` is a default implementation of the IReadWriteSync (1843) interface. It uses a single mutex to protect access to the read/write resource, resulting in a single thread having access to the resource.

See also: IReadWriteSync (1843)

76.101.2 Interfaces overview

Page	Interfaces	Description
1843	IReadWriteSync	Read/Write synchronizer.

76.101.3 Method overview

Page	Method	Description
1897	Beginread	Request read access to the resource.
1897	Beginwrite	Request write access to the resource.
1896	Create	Create a new instance of the `TMultiReadExclusiveWriteSynchronizer` class.
1896	Destroy	Destroys the `TMultiReadExclusiveWriteSynchronizer` instance.
1897	Endread	Release read access to the resource.
1897	Endwrite	Release write access to the resource.

76.101.4 Property overview

Page	Properties	Access	Description
1898	RevisionLevel	r	
1898	WriterThreadID	r	

76.101.5 TMultiReadExclusiveWriteSynchronizer.Create

Synopsis: Create a new instance of the `TMultiReadExclusiveWriteSynchronizer` class.

Declaration: `constructor Create;  Virtual`

Visibility: public

Description: `Create` creates a new instance of `TMultiReadExclusiveWriteSynchronizer`. It initializes a `TRTLCriticalSection`.

Errors: None.

See also: TRTLCriticalSection (1424)

76.101.6 TMultiReadExclusiveWriteSynchronizer.Destroy

Synopsis: Destroys the `TMultiReadExclusiveWriteSynchronizer` instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Create` destroys the instance of `TMultiReadExclusiveWriteSynchronizer`. It frees the `TRTLCriticalSection` it initialized, and calls the inherited destructor.

Errors: None.

See also: TRTLCriticalSection (1424)

76.101.7 TMultiReadExclusiveWriteSynchronizer.Beginwrite

Synopsis: Request write access to the resource.

Declaration: `function Beginwrite : Boolean`

Visibility: public

Description: `Beginwrite` is the implementation of `IReadWriteSync.BeginWrite`. It simply enters the critical section, and returns `True`.

Errors: None.

See also: IReadWriteSync.BeginWrite (1844), EndWrite (1897)

76.101.8 TMultiReadExclusiveWriteSynchronizer.Endwrite

Synopsis: Release write access to the resource.

Declaration: `procedure Endwrite`

Visibility: public

Description: `Beginwrite` is the implementation of `IReadWriteSync.EndWrite`. It simply leaves the critical section.

Errors: None.

See also: IReadWriteSync.EndWrite (1844), BeginWrite (1897)

76.101.9 TMultiReadExclusiveWriteSynchronizer.Beginread

Synopsis: Request read access to the resource.

Declaration: `procedure Beginread`

Visibility: public

Description: `BeginRead` is the implementation of `IReadWriteSync.BeginRead`. It simply attempts to enter the critical section.

Errors: None.

See also: IReadWriteSync.BeginRead (1843), EndRead (1897)

76.101.10 TMultiReadExclusiveWriteSynchronizer.Endread

Synopsis: Release read access to the resource.

Declaration: `procedure Endread`

Visibility: public

Description: `EndRead` is the implementation of `IReadWriteSync.EndRead`. It simply leaves the critical section.

Errors: None.

See also: IReadWriteSync.EndRead (1843), BeginRead (1897)

76.101.11 TMultiReadExclusiveWriteSynchronizer.RevisionLevel

Declaration: `Property RevisionLevel : Cardinal`

Visibility: public

Access: Read

76.101.12 TMultiReadExclusiveWriteSynchronizer.WriterThreadID

Declaration: `Property WriterThreadID : TThreadID`

Visibility: public

Access: Read

76.102 TNativeIntHelper

76.102.1 Description

`TNativeIntHelper` contains some auxiliary routines for a NativeInt-typed ordinal value. It consists mainly of conversion routines to and from other types.

See also: TStringHelper (1926), TShortIntHelper (1910), TSmallIntHelper (1922), TWordHelper (1960), TCardinalHelper (1859), TIntegerHelper (1888), TInt64Helper (1884), TQWordHelper (1906), TNativeIntHelper (1898), TByteHelper (1855)

76.102.2 Method overview

Page	Method	Description
1901	ClearBit	Set bit to 0.
1898	Parse	Convert from a string.
1901	SetBit	Set bit to 1.
1899	Size	Size, in bytes, of the NativeInt value.
1902	TestBit	Check bit.
1900	ToBinString	Convert to a binary string representation.
1900	ToBoolean	Convert to a boolean value.
1900	ToDouble	Convert to a double-sized floating point value.
1900	ToExtended	Convert to an extended-sized floating point value.
1902	ToggleBit	Invert bit.
1901	ToHexString	Convert to a hexadecimal string representation.
1901	ToSingle	Convert to an single-sized floating point value.
1899	ToString	Convert the value to string.
1899	TryParse	Try to convert a string to a NativeInt, report success or failure.

76.102.3 TNativeIntHelper.Parse

Synopsis: Convert from a string.

Declaration: `class function Parse(const AString: string) : NativeInt;  Static`

Visibility: public

Description: `Parse` will attempt to convert the string `AString` to a NativeInt value. It uses the StrToInt (1793) function to perform the conversion, so no localization is taken into account.

Errors: If the string does not contain a valid NativeInt value, an EConvertError (1830) exception is raised.

See also: TNativeIntHelper.ToString (1899), TNativeIntHelper.TryParse (1899), StrToInt (1793)

76.102.4 TNativeIntHelper.Size

Synopsis: Size, in bytes, of the NativeInt value.

Declaration:
```
class function Size : Integer;  Static
```

Visibility: public

Description: `Size` returns the size (in NativeInts) of the NativeInt value. This is equivalent to `SizeOf(NativeInt)`.

Errors: None.

See also: SizeOf (1574)

76.102.5 TNativeIntHelper.ToString

Synopsis: Convert the value to string.

Declaration:
```
class function ToString(const AValue: NativeInt) : string;  Overload
                          ;  Static
function ToString : string;  Overload
```

Visibility: public

Description: `ToString` will, in the class function variant of this method, convert `AValue` to a string representation. In the regular method overloaded version of `ToString`, the NativeInt value itself is used. The IntToStr (1758) function is used to do the conversion.

See also: TNativeIntHelper.Parse (1898), IntToStr (1758)

76.102.6 TNativeIntHelper.TryParse

Synopsis: Try to convert a string to a NativeInt, report success or failure.

Declaration:
```
class function TryParse(const AString: string; out AValue: NativeInt)
                           : Boolean;  Static
```

Visibility: public

Description: `TryParse` attempts to convert the string `AString` to a NativeInt, and reports the success of the attempt. If the attempt is successful, then `True` is returned, and the actual value of the NativeInt is returned in `AValue`.

It uses the val (1635) function to perform the conversion, so no localization is taken into account.

See also: TNativeIntHelper.Parse (1898), Val (1598)

76.102.7 TNativeIntHelper.ToBoolean

Synopsis: Convert to a boolean value.

Declaration: `function ToBoolean : Boolean`

Visibility: public

Description: `ToBoolean` converts the NativeInt value to a boolean: it returns `True` if the value is nonzero, `False` if it is zero.

See also: TNativeIntHelper.ToSingle (1901), TNativeIntHelper.ToDouble (1900), TNativeIntHelper.ToExtended (1900), TNativeIntHelper.ToString (1899), TNativeIntHelper.ToHexString (1901)

76.102.8 TNativeIntHelper.ToDouble

Synopsis: Convert to a double-sized floating point value.

Declaration: `function ToDouble : Double`

Visibility: public

Description: `ToDouble` converts the NativeInt value to a double-sized floating point value.

See also: TNativeIntHelper.ToBoolean (1900), TNativeIntHelper.ToExtended (1900), TNativeIntHelper.ToSingle (1901), TNativeIntHelper.ToString (1899), TNativeIntHelper.ToHexString (1901)

76.102.9 TNativeIntHelper.ToExtended

Synopsis: Convert to an extended-sized floating point value.

Declaration: `function ToExtended : Extended`

Visibility: public

Description: `ToDouble` converts the NativeInt value to an extended-sized floating point value.

See also: TNativeIntHelper.ToBoolean (1900), TNativeIntHelper.ToSingle (1901), TNativeIntHelper.ToDouble (1900), TNativeIntHelper.ToString (1899), TNativeIntHelper.ToHexString (1901)

76.102.10 TNativeIntHelper.ToBinString

Synopsis: Convert to a binary string representation.

Declaration: `function ToBinString : string`

Visibility: public

Description: `ToBinString` converts the NativeInt value to a binary string representation, showing all bits.

See also: TNativeIntHelper.ToHexString (1901), TNativeIntHelper.ToString (1899)

76.102.11 TNativeIntHelper.ToHexString

Synopsis: Convert to a hexadecimal string representation.

Declaration:
```
function ToHexString(const AMinDigits: Integer) : string;  Overload
function ToHexString : string;  Overload
```

Visibility: public

Description: `ToHexString` converts the NativeInt value to a hexadecimal string representation. The `AMinDigits` argument specifies the minimal number of characters in the resulting string. The string will be left-padded with zeroes if the representation contains less than `AMinDigits` characters.

See also: TNativeIntHelper.ToBoolean (1900), TNativeIntHelper.ToSingle (1901), TNativeIntHelper.ToDouble (1900), TNativeIntHelper.ToString (1899), TNativeIntHelper.ToExtended (1900)

76.102.12 TNativeIntHelper.ToSingle

Synopsis: Convert to an single-sized floating point value.

Declaration:
```
function ToSingle : Single
```

Visibility: public

Description: `ToSingle` converts the NativeInt value to a single-sized floating point value.

See also: TNativeIntHelper.ToBoolean (1900), TNativeIntHelper.ToDouble (1900), TNativeIntHelper.ToExtended (1900), TNativeIntHelper.ToString (1899), TNativeIntHelper.ToHexString (1901)

76.102.13 TNativeIntHelper.SetBit

Synopsis: Set bit to 1.

Declaration:
```
function SetBit(const Index: TNativeIntBitIndex) : NativeInt
```

Visibility: public

Description: `SetBit` puts value 1 to bit number determined by argument `Index`.

See also: TNativeIntHelper.ClearBit (1901), TNativeIntHelper.ToggleBit (1902), TNativeIntHelper.TestBit (1902), TNativeIntHelper.HighestSetBitPos (1898), TNativeIntHelper.LowestSetBitPos (1898), TNativeIntHelper.SetBitsCount (1898), TNativeIntHelper.Bits (1898)

76.102.14 TNativeIntHelper.ClearBit

Synopsis: Set bit to 0.

Declaration:
```
function ClearBit(const Index: TNativeIntBitIndex) : NativeInt
```

Visibility: public

Description: `ClearBit` puts value 0 to bit number determined by argument `Index`.

See also: TNativeIntHelper.SetBit (1901), TNativeIntHelper.ToggleBit (1902), TNativeIntHelper.TestBit (1902), TNativeIntHelper.HighestSetBitPos (1898), TNativeIntHelper.LowestSetBitPos (1898), TNativeIntHelper.SetBitsCount (1898), TNativeIntHelper.Bits (1898)

76.102.15 TNativeIntHelper.ToggleBit

Synopsis: Invert bit.

Declaration: `function ToggleBit(const Index: TNativeIntBitIndex) : NativeInt`

Visibility: public

Description: `ToggleBit` reads bit value from bit number determined by argument `Index`, inverts it (if it was 0 it becomes 1, and vice versa), and stores it back.

See also: TNativeIntHelper.SetBit (1901), TNativeIntHelper.ToggleBit (1902), TNativeIntHelper.TestBit (1902), TNativeIntHelper.HighestSetBitPos (1898), TNativeIntHelper.LowestSetBitPos (1898), TNativeIntHelper.SetBitsCount (1898), TNativeIntHelper.Bits (1898)

76.102.16 TNativeIntHelper.TestBit

Synopsis: Check bit.

Declaration: `function TestBit(const Index: TNativeIntBitIndex) : Boolean`

Visibility: public

Description: `TestBit` reads boolean value (true if bit is 1, and false if bit is 0) from bit number determined by argument `Index`.

See also: TNativeIntHelper.SetBit (1901), TNativeIntHelper.ToggleBit (1902), TNativeIntHelper.TestBit (1902), TNativeIntHelper.HighestSetBitPos (1898), TNativeIntHelper.LowestSetBitPos (1898), TNativeIntHelper.SetBitsCount (1898), TNativeIntHelper.Bits (1898)

76.103 TNativeUIntHelper

76.103.1 Description

`NativeUInt` contains some auxiliary routines for a NativeUInt-typed ordinal value. It consists mainly of conversion routines to and from other types.

See also: TStringHelper (1926), TShortIntHelper (1910), TSmallIntHelper (1922), TWordHelper (1960), TCardinalHelper (1859), TIntegerHelper (1888), TInt64Helper (1884), TQWordHelper (1906), TNativeIntHelper (1898), TByteHelper (1855)

76.103.2 Method overview

Page	Method	Description
1906	ClearBit	Set bit to 0.
1903	Parse	Convert from a string.
1905	SetBit	Set bit to 1.
1903	Size	Size, in bytes, of the NativeUInt value.
1906	TestBit	Check bit.
1905	ToBinString	Convert to a binary string representation.
1904	ToBoolean	Convert to a boolean value.
1904	ToDouble	Convert to a double-sized floating point value.
1904	ToExtended	Convert to an extended-sized floating point value.
1906	ToggleBit	Invert bit.
1905	ToHexString	Convert to a hexadecimal string representation.
1905	ToSingle	Convert to an single-sized floating point value.
1903	ToString	Convert the value to string.
1904	TryParse	Try to convert a string to a NativeUInt, report success or failure.

76.103.3 TNativeUIntHelper.Parse

Synopsis: Convert from a string.

Declaration:
```
class function Parse(const AString: string) : NativeUInt;  Static
```

Visibility: public

Description: `Parse` will attempt to convert the string `AString` to a NativeUInt value. It uses the StrToInt (1793) function to perform the conversion, so no localization is taken into account.

Errors: If the string does not contain a valid NativeUInt value, an EConvertError (1830) exception is raised.

See also: TNativeUIntHelper.ToString (1903), TNativeUIntHelper.TryParse (1904), StrToInt (1793)

76.103.4 TNativeUIntHelper.Size

Synopsis: Size, in bytes, of the NativeUInt value.

Declaration:
```
class function Size : Integer;  Static
```

Visibility: public

Description: `Size` returns the size (in NativeUInts) of the NativeUInt value. This is equivalent to `SizeOf(NativeUInt)`.

Errors: None.

See also: SizeOf (1574)

76.103.5 TNativeUIntHelper.ToString

Synopsis: Convert the value to string.

Declaration:
```
class function ToString(const AValue: NativeUInt) : string;  Overload
                              ;  Static
function ToString : string;  Overload
```

Visibility: public

Description: `ToString` will, in the class function variant of this method, convert `AValue` to a string representation. In the regular method overloaded version of `ToString`, the NativeUInt value itself is used. The IntToStr (1758) function is used to do the conversion.

See also: TNativeUIntHelper.Parse (1903), IntToStr (1758)

76.103.6 TNativeUIntHelper.TryParse

Synopsis: Try to convert a string to a NativeUInt, report success or failure.

Declaration:
```
class function TryParse(const AString: string; out AValue: NativeUInt)
                        : Boolean;  Static
```

Visibility: public

Description: `TryParse` attempts to convert the string `AString` to a NativeUInt, and reports the success of the attempt. If the attempt is successful, then `True` is returned, and the actual value of the NativeUInt is returned in `AValue`.

It uses the val (1635) function to perform the conversion, so no localization is taken into account.

See also: TNativeUIntHelper.Parse (1903), Val (1598)

76.103.7 TNativeUIntHelper.ToBoolean

Synopsis: Convert to a boolean value.

Declaration:
```
function ToBoolean : Boolean
```

Visibility: public

Description: `ToBoolean` converts the NativeUInt value to a boolean: it returns `True` if the value is nonzero, `False` if it is zero.

See also: TNativeUIntHelper.ToSingle (1905), TNativeUIntHelper.ToDouble (1904), TNativeUIntHelper.ToExtended (1904), TNativeUIntHelper.ToString (1903), TNativeUIntHelper.ToHexString (1905)

76.103.8 TNativeUIntHelper.ToDouble

Synopsis: Convert to a double-sized floating point value.

Declaration:
```
function ToDouble : Double
```

Visibility: public

Description: `ToDouble` converts the NativeUInt value to a double-sized floating point value.

See also: TNativeUIntHelper.ToBoolean (1904), TNativeUIntHelper.ToExtended (1904), TNativeUIntHelper.ToSingle (1905), TNativeUIntHelper.ToString (1903), TNativeUIntHelper.ToHexString (1905)

76.103.9 TNativeUIntHelper.ToExtended

Synopsis: Convert to an extended-sized floating point value.

Declaration:
```
function ToExtended : Extended
```

Visibility: public

Description: `ToDouble` converts the NativeUInt value to an extended-sized floating point value.

See also: TNativeUIntHelper.ToBoolean (1904), TNativeUIntHelper.ToSingle (1905), TNativeUIntHelper.ToDouble (1904), TNativeUIntHelper.ToString (1903), TNativeUIntHelper.ToHexString (1905)

76.103.10 TNativeUIntHelper.ToBinString

Synopsis: Convert to a binary string representation.

Declaration: `function ToBinString : string`

Visibility: public

Description: `ToBinString` converts the NativeUInt value to a binary string representation, showing all bits.

76.103.11 TNativeUIntHelper.ToHexString

Synopsis: Convert to a hexadecimal string representation.

Declaration:
```
function ToHexString(const AMinDigits: Integer) : string;  Overload
function ToHexString : string;  Overload
```

Visibility: public

Description: `ToHexString` converts the NativeUInt value to a hexadecimal string representation. The `AMinDigits` argument specifies the minimal number of characters in the resulting string. The string will be left-padded with zeroes if the representation contains less than `AMinDigits` characters.

See also: TNativeUIntHelper.ToBoolean (1904), TNativeUIntHelper.ToSingle (1905), TNativeUIntHelper.ToDouble (1904), TNativeUIntHelper.ToString (1903), TNativeUIntHelper.ToExtended (1904)

76.103.12 TNativeUIntHelper.ToSingle

Synopsis: Convert to an single-sized floating point value.

Declaration: `function ToSingle : Single`

Visibility: public

Description: `ToSingle` converts the NativeUInt value to a single-sized floating point value.

See also: TNativeUIntHelper.ToBoolean (1904), TNativeUIntHelper.ToDouble (1904), TNativeUIntHelper.ToExtended (1904), TNativeUIntHelper.ToString (1903), TNativeUIntHelper.ToHexString (1905)

76.103.13 TNativeUIntHelper.SetBit

Synopsis: Set bit to 1.

Declaration: `function SetBit(const Index: TNativeUIntBitIndex) : NativeUInt`

Visibility: public

Description: `SetBit` puts value 1 to bit number determined by argument `Index`.

See also: TNativeUIntHelper.ClearBit (1906), TNativeUIntHelper.ToggleBit (1906), TNativeUIntHelper.TestBit (1906), TNativeUIntHelper.HighestSetBitPos (1902), TNativeUIntHelper.LowestSetBitPos (1902), TNativeUIntHelper.SetBitsCount (1902), TNativeUIntHelper.Bits (1902)

76.103.14 TNativeUIntHelper.ClearBit

Synopsis: Set bit to 0.

Declaration: `function ClearBit(const Index: TNativeUIntBitIndex) : NativeUInt`

Visibility: public

Description: `ClearBit` puts value 0 to bit number determined by argument `Index`.

See also: TNativeUIntHelper.SetBit (1905), TNativeUIntHelper.ToggleBit (1906), TNativeUIntHelper.TestBit (1906), TNativeUIntHelper.HighestSetBitPos (1902), TNativeUIntHelper.LowestSetBitPos (1902), TNativeUIntHelper.SetBitsCount (1902), TNativeUIntHelper.Bits (1902)

76.103.15 TNativeUIntHelper.ToggleBit

Synopsis: Invert bit.

Declaration: `function ToggleBit(const Index: TNativeUIntBitIndex) : NativeUInt`

Visibility: public

Description: `ToggleBit` reads bit value from bit number determined by argument `Index`, inverts it (if it was 0 it becomes 1, and vice versa), and stores it back.

See also: TNativeUIntHelper.SetBit (1905), TNativeUIntHelper.ToggleBit (1906), TNativeUIntHelper.TestBit (1906), TNativeUIntHelper.HighestSetBitPos (1902), TNativeUIntHelper.LowestSetBitPos (1902), TNativeUIntHelper.SetBitsCount (1902), TNativeUIntHelper.Bits (1902)

76.103.16 TNativeUIntHelper.TestBit

Synopsis: Check bit.

Declaration: `function TestBit(const Index: TNativeUIntBitIndex) : Boolean`

Visibility: public

Description: `TestBit` reads boolean value (true if bit is 1, and false if bit is 0) from bit number determined by argument `Index`.

See also: TNativeUIntHelper.SetBit (1905), TNativeUIntHelper.ToggleBit (1906), TNativeUIntHelper.TestBit (1906), TNativeUIntHelper.HighestSetBitPos (1902), TNativeUIntHelper.LowestSetBitPos (1902), TNativeUIntHelper.SetBitsCount (1902), TNativeUIntHelper.Bits (1902)

76.104 TQWordHelper

76.104.1 Description

`TQWordHelper` contains some auxiliary routines for a QWord-typed ordinal value. It consists mainly of conversion routines to and from other types.

See also: TStringHelper (1926), TShortIntHelper (1910), TSmallIntHelper (1922), TWordHelper (1960), TCardinalHelper (1859), TIntegerHelper (1888), TInt64Helper (1884), TByteHelper (1855), TNativeIntHelper (1898), TNativeUIntHelper (1902)

76.104.2 Method overview

Page	Method	Description
1910	ClearBit	Set bit to 0.
1907	Parse	Convert from a string.
1909	SetBit	Set bit to 1.
1907	Size	Size, in bytes, of the QWord value.
1910	TestBit	Check bit.
1909	ToBinString	Convert to a binary string representation.
1908	ToBoolean	Convert to a boolean value.
1908	ToDouble	Convert to a double-sized floating point value.
1908	ToExtended	Convert to an extended-sized floating point value.
1910	ToggleBit	Invert bit.
1909	ToHexString	Convert to a hexadecimal string representation.
1909	ToSingle	Convert to an single-sized floating point value.
1907	ToString	Convert the value to string.
1908	TryParse	Try to convert a string to a QWord, report success or failure.

76.104.3 TQWordHelper.Parse

Synopsis: Convert from a string.

Declaration:
```
class function Parse(const AString: string) : QWord;  Static
```

Visibility: public

Description: `Parse` will attempt to convert the string `AString` to a QWord value. It uses the StrToQWord (1795) function to perform the conversion, so no localization is taken into account.

Errors: If the string does not contain a valid QWord value, an EConvertError (1830) exception is raised.

See also: TQWordHelper.ToString (1907), TQWordHelper.TryParse (1908), StrToQWord (1795)

76.104.4 TQWordHelper.Size

Synopsis: Size, in bytes, of the QWord value.

Declaration:
```
class function Size : Integer;  Static
```

Visibility: public

Description: `Size` returns the size (in QWords) of the QWord value. This is equivalent to `SizeOf(QWord)`.

Errors: None.

See also: SizeOf (1574)

76.104.5 TQWordHelper.ToString

Synopsis: Convert the value to string.

Declaration:
```
class function ToString(const AValue: QWord) : string;  Overload
                              ;  Static
function ToString : string;  Overload
```

Visibility: public

Description: `ToString` will, in the class function variant of this method, convert `AValue` to a string representation. In the regular method overloaded version of `ToString`, the QWord value itself is used. The IntToStr (1758) function is used to do the conversion.

See also: TQWordHelper.Parse (1907), IntToStr (1758)

76.104.6 TQWordHelper.TryParse

Synopsis: Try to convert a string to a QWord, report success or failure.

Declaration:
```
class function TryParse(const AString: string; out AValue: QWord)
                        : Boolean;  Static
```

Visibility: public

Description: `TryParse` attempts to convert the string `AString` to a QWord, and reports the success of the attempt. If the attempt is successful, then `True` is returned, and the actual value of the QWord is returned in `AValue`.

It uses the val (1635) function to perform the conversion, so no localization is taken into account.

See also: TQWordHelper.Parse (1907), Val (1598)

76.104.7 TQWordHelper.ToBoolean

Synopsis: Convert to a boolean value.

Declaration:
```
function ToBoolean : Boolean
```

Visibility: public

Description: `ToBoolean` converts the QWord value to a boolean: it returns `True` if the value is nonzero, `False` if it is zero.

See also: TQWordHelper.ToSingle (1909), TQWordHelper.ToDouble (1908), TQWordHelper.ToExtended (1908), TQWordHelper.ToString (1907), TQWordHelper.ToHexString (1909)

76.104.8 TQWordHelper.ToDouble

Synopsis: Convert to a double-sized floating point value.

Declaration:
```
function ToDouble : Double
```

Visibility: public

Description: `ToDouble` converts the QWord value to a double-sized floating point value.

See also: TQWordHelper.ToBoolean (1908), TQWordHelper.ToExtended (1908), TQWordHelper.ToSingle (1909), TQWordHelper.ToString (1907), TQWordHelper.ToHexString (1909)

76.104.9 TQWordHelper.ToExtended

Synopsis: Convert to an extended-sized floating point value.

Declaration:
```
function ToExtended : Extended
```

Visibility: public

Description: `ToDouble` converts the QWord value to an extended-sized floating point value.

See also: TQWordHelper.ToBoolean (1908), TQWordHelper.ToSingle (1909), TQWordHelper.ToDouble (1908), TQWordHelper.ToString (1907), TQWordHelper.ToHexString (1909)

76.104.10 TQWordHelper.ToBinString

Synopsis: Convert to a binary string representation.

Declaration: `function ToBinString : string`

Visibility: public

Description: `ToBinString` converts the QWord value to a binary string representation, showing all 64 bits.

See also: TQWordHelper.ToHexString (1909), TQWordHelper.ToString (1907)

76.104.11 TQWordHelper.ToHexString

Synopsis: Convert to a hexadecimal string representation.

Declaration:
```
function ToHexString(const AMinDigits: Integer) : string;  Overload
function ToHexString : string;  Overload
```

Visibility: public

Description: `ToHexString` converts the QWord value to a hexadecimal string representation. The `AMinDigits` argument specifies the minimal number of characters in the resulting string. The string will be left-padded with zeroes if the representation contains less than `AMinDigits` characters.

See also: TQWordHelper.ToBoolean (1908), TQWordHelper.ToSingle (1909), TQWordHelper.ToDouble (1908), TQWordHelper.ToString (1907), TQWordHelper.ToExtended (1908)

76.104.12 TQWordHelper.ToSingle

Synopsis: Convert to an single-sized floating point value.

Declaration: `function ToSingle : Single`

Visibility: public

Description: `ToSingle` converts the QWord value to a single-sized floating point value.

See also: TQWordHelper.ToBoolean (1908), TQWordHelper.ToDouble (1908), TQWordHelper.ToExtended (1908), TQWordHelper.ToString (1907), TQWordHelper.ToHexString (1909)

76.104.13 TQWordHelper.SetBit

Synopsis: Set bit to 1.

Declaration: `function SetBit(const Index: TQwordBitIndex) : QWord`

Visibility: public

Description: `SetBit` puts value 1 to bit number determined by argument `Index`.

See also: TQWordHelper.ClearBit (1910), TQWordHelper.ToggleBit (1910), TQWordHelper.TestBit (1910), TQWordHelper.HighestSetBitPos (1906), TQWordHelper.LowestSetBitPos (1906), TQWordHelper.SetBitsCount (1906), TQWordHelper.Bits (1906)

76.104.14 TQWordHelper.ClearBit

Synopsis: Set bit to 0.

Declaration: `function ClearBit(const Index: TQwordBitIndex) : QWord`

Visibility: public

Description: `ClearBit` puts value 0 to bit number determined by argument `Index`.

See also: TQWordHelper.SetBit (1909), TQWordHelper.ToggleBit (1910), TQWordHelper.TestBit (1910), TQWordHelper.HighestSetBitPos (1906), TQWordHelper.LowestSetBitPos (1906), TQWordHelper.SetBitsCount (1906), TQWordHelper.Bits (1906)

76.104.15 TQWordHelper.ToggleBit

Synopsis: Invert bit.

Declaration: `function ToggleBit(const Index: TQwordBitIndex) : QWord`

Visibility: public

Description: `ToggleBit` reads bit value from bit number determined by argument `Index`, inverts it (if it was 0 it becomes 1, and vice versa), and stores it back.

See also: TQWordHelper.SetBit (1909), TQWordHelper.ToggleBit (1910), TQWordHelper.TestBit (1910), TQWordHelper.HighestSetBitPos (1906), TQWordHelper.LowestSetBitPos (1906), TQWordHelper.SetBitsCount (1906), TQWordHelper.Bits (1906)

76.104.16 TQWordHelper.TestBit

Synopsis: Check bit.

Declaration: `function TestBit(const Index: TQwordBitIndex) : Boolean`

Visibility: public

Description: `TestBit` reads boolean value (true if bit is 1, and false if bit is 0) from bit number determined by argument `Index`.

See also: TQWordHelper.SetBit (1909), TQWordHelper.ToggleBit (1910), TQWordHelper.TestBit (1910), TQWordHelper.HighestSetBitPos (1906), TQWordHelper.LowestSetBitPos (1906), TQWordHelper.SetBitsCount (1906), TQWordHelper.Bits (1906)

76.105 TShortIntHelper

76.105.1 Description

`TShortIntHelper` contains some auxiliary routines for a ShortInt-typed ordinal value. It consists mainly of conversion routines to and from other types.

See also: TStringHelper (1926), TByteHelper (1855), TSmallIntHelper (1922), TWordHelper (1960), TCardinalHelper (1859), TIntegerHelper (1888), TInt64Helper (1884), TQWordHelper (1906), TNativeIntHelper (1898), TNativeUIntHelper (1902)

76.105.2 Method overview

Page	Method	Description
1914	ClearBit	Set bit to 0.
1911	Parse	Convert from a string.
1913	SetBit	Set bit to 1.
1911	Size	Size, in bytes, of the ShortInt value.
1914	TestBit	Check bit.
1913	ToBinString	Convert to a binary string representation.
1912	ToBoolean	Convert to a boolean value.
1912	ToDouble	Convert to a double-sized floating point value.
1912	ToExtended	Convert to an extended-sized floating point value.
1914	ToggleBit	Invert bit.
1913	ToHexString	Convert to a hexadecimal string representation.
1913	ToSingle	Convert to an single-sized floating point value.
1911	ToString	Convert the value to string.
1912	TryParse	Try to convert a string to a ShortInt, report success or failure.

76.105.3 TShortIntHelper.Parse

Synopsis: Convert from a string.

Declaration:
```
class function Parse(const AString: string) : ShortInt;  Static
```

Visibility: public

Description: `Parse` will attempt to convert the string `AString` to a ShortInt value. It uses the StrToInt (1793) function to perform the conversion, so no localization is taken into account.

Errors: If the string does not contain a valid ShortInt value, an EConvertError (1830) exception is raised.

See also: TShortIntHelper.ToString (1911), TShortIntHelper.TryParse (1912), StrToInt (1793)

76.105.4 TShortIntHelper.Size

Synopsis: Size, in bytes, of the ShortInt value.

Declaration:
```
class function Size : Integer;  Static
```

Visibility: public

Description: `Size` returns the size (in ShortInts) of the ShortInt value. This is equivalent to `SizeOf(Byte)`.

Errors: None.

See also: SizeOf (1574)

76.105.5 TShortIntHelper.ToString

Synopsis: Convert the value to string.

Declaration:
```
class function ToString(const AValue: ShortInt) : string;  Overload
                              ;  Static
function ToString : string;  Overload
```

Visibility: public

Description: `ToString` will, in the class function variant of this method, convert `AValue` to a string representation. In the regular method overloaded version of `ToString`, the ShortInt value itself is used. The IntToStr (1758) function is used to do the conversion.

See also: TShortIntHelper.Parse (1911), IntToStr (1758)

76.105.6 TShortIntHelper.TryParse

Synopsis: Try to convert a string to a ShortInt, report success or failure.

Declaration:
```
class function TryParse(const AString: string; out AValue: ShortInt)
                        : Boolean;  Static
```

Visibility: public

Description: `TryParse` attempts to convert the string `AString` to a ShortInt, and reports the success of the attempt. If the attempt is successful, then `True` is returned, and the actual value of the ShortInt is returned in `AValue`.

It uses the val (1635) function to perform the conversion, so no localization is taken into account.

See also: TShortIntHelper.Parse (1911), Val (1598)

76.105.7 TShortIntHelper.ToBoolean

Synopsis: Convert to a boolean value.

Declaration:
```
function ToBoolean : Boolean
```

Visibility: public

Description: `ToBoolean` converts the ShortInt value to a boolean: it returns `True` if the value is nonzero, `False` if it is zero.

See also: TShortIntHelper.ToSingle (1913), TShortIntHelper.ToDouble (1912), TShortIntHelper.ToExtended (1912), TShortIntHelper.ToString (1911), TShortIntHelper.ToHexString (1913)

76.105.8 TShortIntHelper.ToDouble

Synopsis: Convert to a double-sized floating point value.

Declaration:
```
function ToDouble : Double
```

Visibility: public

Description: `ToDouble` converts the ShortInt value to a double-sized floating point value.

See also: TShortIntHelper.ToBoolean (1912), TShortIntHelper.ToExtended (1912), TShortIntHelper.ToSingle (1913), TShortIntHelper.ToString (1911), TShortIntHelper.ToHexString (1913)

76.105.9 TShortIntHelper.ToExtended

Synopsis: Convert to an extended-sized floating point value.

Declaration:
```
function ToExtended : Extended
```

Visibility: public

Description: `ToDouble` converts the ShortInt value to an extended-sized floating point value.

See also: TShortIntHelper.ToBoolean (1912), TShortIntHelper.ToSingle (1913), TShortIntHelper.ToDouble (1912), TShortIntHelper.ToString (1911), TShortIntHelper.ToHexString (1913)

76.105.10 TShortIntHelper.ToBinString

Synopsis: Convert to a binary string representation.

Declaration: `function ToBinString : string`

Visibility: public

Description: `ToBinString` converts the shortint value to a binary string representation, showing all 8 bits.

See also: TShortIntHelper.ToHexString (1913), TShortIntHelper.ToString (1911)

76.105.11 TShortIntHelper.ToHexString

Synopsis: Convert to a hexadecimal string representation.

Declaration:
```
function ToHexString(const AMinDigits: Integer) : string;  Overload
function ToHexString : string;  Overload
```

Visibility: public

Description: `ToHexString` converts the ShortInt value to a hexadecimal string representation. The `AMinDigits` argument specifies the minimal number of characters in the resulting string. The string will be left-padded with zeroes if the representation contains less than `AMinDigits` characters.

See also: TShortIntHelper.ToBoolean (1912), TShortIntHelper.ToSingle (1913), TShortIntHelper.ToDouble (1912), TShortIntHelper.ToString (1911), TShortIntHelper.ToExtended (1912)

76.105.12 TShortIntHelper.ToSingle

Synopsis: Convert to an single-sized floating point value.

Declaration: `function ToSingle : Single`

Visibility: public

Description: `ToSingle` converts the ShortInt value to a single-sized floating point value.

See also: TShortIntHelper.ToBoolean (1912), TShortIntHelper.ToDouble (1912), TShortIntHelper.ToExtended (1912), TShortIntHelper.ToString (1911), TShortIntHelper.ToHexString (1913)

76.105.13 TShortIntHelper.SetBit

Synopsis: Set bit to 1.

Declaration: `function SetBit(const Index: TShortIntBitIndex) : ShortInt`

Visibility: public

Description: `SetBit` puts value 1 to bit number determined by argument `Index`.

See also: TShortIntHelper.ClearBit (1914), TShortIntHelper.ToggleBit (1914), TShortIntHelper.TestBit (1914), TShortIntHelper.HighestSetBitPos (1910), TShortIntHelper.LowestSetBitPos (1910), TShortIntHelper.SetBitsCount (1910), TShortIntHelper.Bits (1910)

76.105.14 TShortIntHelper.ClearBit

Synopsis: Set bit to 0.

Declaration: `function ClearBit(const Index: TShortIntBitIndex) : ShortInt`

Visibility: public

Description: `ClearBit` puts value 0 to bit number determined by argument `Index`.

See also: TShortIntHelper.SetBit (1913), TShortIntHelper.ToggleBit (1914), TShortIntHelper.TestBit (1914), TShortIntHelper.HighestSetBitPos (1910), TShortIntHelper.LowestSetBitPos (1910), TShortIntHelper.SetBitsCount (1910), TShortIntHelper.Bits (1910)

76.105.15 TShortIntHelper.ToggleBit

Synopsis: Invert bit.

Declaration: `function ToggleBit(const Index: TShortIntBitIndex) : ShortInt`

Visibility: public

Description: `ToggleBit` reads bit value from bit number determined by argument `Index`, inverts it (if it was 0 it becomes 1, and vice versa), and stores it back.

See also: TShortIntHelper.SetBit (1913), TShortIntHelper.ClearBit (1914), TShortIntHelper.TestBit (1914), TShortIntHelper.HighestSetBitPos (1910), TShortIntHelper.LowestSetBitPos (1910), TShortIntHelper.SetBitsCount (1910), TShortIntHelper.Bits (1910)

76.105.16 TShortIntHelper.TestBit

Synopsis: Check bit.

Declaration: `function TestBit(const Index: TShortIntBitIndex) : Boolean`

Visibility: public

Description: `TestBit` reads boolean value (true if bit is 1, and false if bit is 0) from bit number determined by argument `Index`.

See also: TShortIntHelper.SetBit (1913), TShortIntHelper.ClearBit (1914), TShortIntHelper.ToggleBit (1914), TShortIntHelper.HighestSetBitPos (1910), TShortIntHelper.LowestSetBitPos (1910), TShortIntHelper.SetBitsCount (1910), TShortIntHelper.Bits (1910)

76.106 TSimpleRWSync

76.106.1 Description

`TSimpleRWSync` implements a simple read/write locking mechanism. It controls access to an object: only a single thread is allowed access to an object for either read or write operations.

Access is controlled through a single critical section.

See also: TMultiReadExclusiveWriteSynchronizer (1895)

76.106.2 Interfaces overview

Page	Interfaces	Description
1843	IReadWriteSync	Read/Write synchronizer.

76.106.3 Method overview

Page	Method	Description
1916	Beginread	Acquire a read lock.
1915	Beginwrite	Acquire a write lock.
1915	Create	Create a new instance of `TSimpleRWSync`.
1915	Destroy	Removes the `TSimpleRWSync` instance from memory.
1916	Endread	Release the read lock.
1916	Endwrite	Release the write lock.

76.106.4 TSimpleRWSync.Create

Synopsis: Create a new instance of `TSimpleRWSync`.

Declaration: `constructor Create;  Virtual`

Visibility: public

Description: `Create` allocates a new instance of `TSimpleRWSync` and initializes the critical section for use in the various methods.

See also: TSimpleRWSync.Destroy (1915)

76.106.5 TSimpleRWSync.Destroy

Synopsis: Removes the `TSimpleRWSync` instance from memory.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` releases the critical section and removes the `TSimpleRWSync` instance from memory.

See also: TSimpleRWSync.Create (1915)

76.106.6 TSimpleRWSync.Beginwrite

Synopsis: Acquire a write lock.

Declaration: `function Beginwrite : Boolean`

Visibility: public

Description: `Beginwrite` waits till all other threads have released their read or write locks on the object, and then acquires a write lock on the object.

See also: TSimpleRWSync.BeginRead (1916), TSimpleRWSync.EndWrite (1916)

76.106.7 TSimpleRWSync.Endwrite

Synopsis: Release the write lock.

Declaration: `procedure Endwrite`

Visibility: public

Description: `EndWrite` releases the current threads lock on the object, allowing other threads to acquire a read or write lock on the object.

See also: TSimpleRWSync.BeginWrite (1915), TSimpleRWSync.EndRead (1916)

76.106.8 TSimpleRWSync.Beginread

Synopsis: Acquire a read lock.

Declaration: `procedure Beginread`

Visibility: public

Description: `BeginRead` waits till all other threads have released their read or write locks on the object, and then acquires a read lock on the object.

See also: TSimpleRWSync.BeginWrite (1915), TSimpleRWSync.EndRead (1916)

76.106.9 TSimpleRWSync.Endread

Synopsis: Release the read lock.

Declaration: `procedure Endread`

Visibility: public

Description: `EndRead` releases the current threads read lock on the object, allowing other threads to acquire a read or write lock on the object.

See also: TSimpleRWSync.BeginRead (1916), TSimpleRWSync.EndWrite (1916)

76.107 TSingleHelper

76.107.1 Description

`TSingleHelper` is the helper type for the single-sized floating point type. It contains some conversion methods, as well as access to the low-level structure of the floating-point representation of a single.

See also: TDoubleHelper (1863), TExtendedHelper (1876)

76.107.2 Method overview

Page	Method	Description
1920	BuildUp	Build a single-sized floating point from its composing parts.
1920	Exponent	Exponent of the floating-point value.
1920	Fraction	Fraction of the floating-point value.
1917	IsInfinity	Check whether a value is positive or negative infinity.
1917	IsNan	Check whether a value equals `NaN`.
1918	IsNegativeInfinity	Check whether a value is negative infinity.
1918	IsPositiveInfinity	Check whether a value is positive infinity.
1921	Mantissa	Mantissa of the floating-point.
1918	Parse	Convert a string to a floating point value.
1919	Size	Size (in bytes) of a single-sized floating point value.
1921	SpecialType	Return the type of the single-sized floating point value.
1919	ToString	Convert a single-sized floating point value to a string.
1920	TryParse	Try to convert a string to a single-sized floating point value.

76.107.3 Property overview

Page	Properties	Access	Description
1921	Bytes	rw	Indexed access to the individual bytes of the floating point value.
1922	Exp	rw	The bit pattern of the exponent as stored in memory.
1922	Frac	rw	Bitpattern that makes up the fractional part.
1922	Sign	rw	Sign of the floating point value.
1921	Words	rw	Indexed access to the words that make up the floating point value.

76.107.4 TSingleHelper.IsNan

Synopsis: Check whether a value equals `NaN`.

Declaration:
```
class function IsNan(const AValue: Single) : Boolean;  Overload;  Static
function IsNan : Boolean;  Overload
```

Visibility: public

Description: `IsNan` checks whether a single-sized floating point value is `NaN` (Not a Number). If so, it returns `True`. When the class function version is used, the value can be specified using `AValue`. In the method version, the used value is (`Self`).

See also: TSingleHelper.IsInfinity (1917), TSingleHelper.IsPositiveInfinity (1918), TSingleHelper.IsNegativeInfinity (1918)

76.107.5 TSingleHelper.IsInfinity

Synopsis: Check whether a value is positive or negative infinity.

Declaration:
```
class function IsInfinity(const AValue: Single) : Boolean;  Overload
                          ;  Static
function IsInfinity : Boolean;  Overload
```

Visibility: public

Description: `IsInfinity` checks whether a single-sized floating point value represents a positive or negative infinity. If so, it returns `True`. When the class function version is used, the value can be specified using `AValue`. In the method version, the used value is (`Self`).

See also: TSingleHelper.IsNan (1917), TSingleHelper.IsPositiveInfinity (1918), TSingleHelper.IsNegativeInfinity (1918)

76.107.6 TSingleHelper.IsNegativeInfinity

Synopsis: Check whether a value is negative infinity.

Declaration:
```
class function IsNegativeInfinity(const AValue: Single) : Boolean
                                  ;  Overload;  Static
function IsNegativeInfinity : Boolean;  Overload
```

Visibility: public

Description: `IsNegativeInfinity` checks whether a single-sized floating point value represents a negative infinity. If so, it returns `True`. When the class function version is used, the value can be specified using `AValue`. In the method version, the used value is (`Self`).

See also: TSingleHelper.IsNan (1917), TSingleHelper.IsPositiveInfinity (1918), TSingleHelper.IsInfinity (1917)

76.107.7 TSingleHelper.IsPositiveInfinity

Synopsis: Check whether a value is positive infinity.

Declaration:
```
class function IsPositiveInfinity(const AValue: Single) : Boolean
                                  ;  Overload;  Static
function IsPositiveInfinity : Boolean;  Overload
```

Visibility: public

Description: `IsPositiveInfinity` checks whether a single-sized floating point value represents a positive infinity. If so, it returns `True`. When the class function version is used, the value can be specified using `AValue`. In the method version, the used value is (`Self`).

See also: TSingleHelper.IsNan (1917), TSingleHelper.IsNegativeInfinity (1918), TSingleHelper.IsInfinity (1917)

76.107.8 TSingleHelper.Parse

Synopsis: Convert a string to a floating point value.

Declaration:
```
class function Parse(const AString: string) : Single;  Overload;  Static
class function Parse(const AString: string;
                     const AFormatSettings: TFormatSettings) : Single
                     ;  Overload;  Static
```

Visibility: public

Description: `Parse` will try to convert `AString` to a single-sized floating point value. It will take into account internationalization settings. (it uses `FloatToStr`).

Errors: If the string `AString` is not a valid floating-point value, a EConvertError (1830) exception is raised.

See also: FloatToStr (1729), TSingleHelper.ToString (1919), TSingleHelper.TryParse (1920)

76.107.9 TSingleHelper.Size

Synopsis: Size (in bytes) of a single-sized floating point value.

Declaration:
```
class function Size : Integer;  Static
```

Visibility: public

Description: `Size` is the size (in bytes) of a single-sized floating point value. It is equivalent to calling `SizeOf(Single)`.

See also: SizeOf (1574)

76.107.10 TSingleHelper.ToString

Synopsis: Convert a single-sized floating point value to a string.

Declaration:
```
class function ToString(const AValue: Single) : string;  Overload
                        ;  Static
class function ToString(const AValue: Single;
                        const AFormatSettings: TFormatSettings) : string
                        ;  Overload;  Static
class function ToString(const AValue: Single;
                        const AFormat: TFloatFormat;
                        const APrecision: Integer;
                        const ADigits: Integer) : string;  Overload
                        ;  Static
class function ToString(const AValue: Single;
                        const AFormat: TFloatFormat;
                        const APrecision: Integer;
                        const ADigits: Integer;
                        const AFormatSettings: TFormatSettings) : string
                        ;  Overload;  Static
function ToString(const AFormat: TFloatFormat;
                  const APrecision: Integer; const ADigits: Integer)
                   : string;  Overload
function ToString(const AFormat: TFloatFormat;
                  const APrecision: Integer; const ADigits: Integer;
                  const AFormatSettings: TFormatSettings) : string
                  ;  Overload
function ToString(const AFormatSettings: TFormatSettings) : string
                  ;  Overload
function ToString : string;  Overload
```

Visibility: public

Description: `ToString` will convert `AValue` (or `Self` in the plain method version) to a string. Optionally `FormatSettings` can be specified, to be able to specify the decimal separator character to use.

Additionally, a precision `APrecision` and number of digits `ADigits` can be specified, in conjunction with a `AFormat` parameter to specify the form in which the floating-point value must be represented. (see TFloatFormat (1656) for an explanation of the various values). In this case, FloatToStrF (1731) is used to format the value. In the absence of these parameters, FloatToStr (1729) is called.

See also: FloatToStr (1729), FloatToStrF (1731), TFloatFormat (1656)

76.107.11 TSingleHelper.TryParse

Synopsis: Try to convert a string to a single-sized floating point value.

Declaration:
```
class function TryParse(const AString: string; out AValue: Single)
                         : Boolean;  Overload;  Static
class function TryParse(const AString: string; out AValue: Single;
                        const AFormatSettings: TFormatSettings) : Boolean
                        ;  Overload;  Static
```

Visibility: public

Description: `TryParse` attempts to convert the string `AString` to a single-sized floating point value and reports `True` if the conversion was successful. In that case the parsed value is returned in `AValue`.

If the conversion failed, `False` is returned.

See also: TSingleHelper.Parse (1918), TSingleHelper.ToString (1919)

76.107.12 TSingleHelper.BuildUp

Synopsis: Build a single-sized floating point from its composing parts.

Declaration:
```
procedure BuildUp(const ASignFlag: Boolean; const AMantissa: QWord;
                  const AExponent: Integer)
```

Visibility: public

Description: `BuildUp` will compose a single-sized floating point value from the sign `ASignFlag`, mantissa `AMantissa` and exponent `AExponent`. It simply sets the Sign (1922), Exp (1922) and Frac (1922) properties in 1 call.

See also: TSingleHelper.Sign (1922), TSingleHelper.Exp (1922), TSingleHelper.Frac (1922)

76.107.13 TSingleHelper.Exponent

Synopsis: Exponent of the floating-point value.

Declaration:
```
function Exponent : Integer
```

Visibility: public

Description: `Exponent` is the value `X` in the representation of the floating-point value in $m*2\hat{X}$, i.e. the exponent.

See also: TSingleHelper.Sign (1922), TSingleHelper.Exp (1922), TSingleHelper.Frac (1922), TSingleHelper.Fraction (1920), TSingleHelper.Mantissa (1921)

76.107.14 TSingleHelper.Fraction

Synopsis: Fraction of the floating-point value.

Declaration:
```
function Fraction : Extended
```

Visibility: public

Description: `Fraction` is the decimal part of the floating-point value.

See also: TSingleHelper.Sign (1922), TSingleHelper.Exp (1922), TSingleHelper.Exponent (1920), TSingleHelper.Frac (1922), TSingleHelper.Mantissa (1921)

76.107.15 TSingleHelper.Mantissa

Synopsis: Mantissa of the floating-point.

Declaration: `function Mantissa : QWord`

Visibility: public

Description: `Mantissa` is the value of the significant without the hidden bit. This means it the plain bit pattern as it is stored in memory.

See also: TSingleHelper.Sign (1922), TSingleHelper.Exp (1922), TSingleHelper.Exponent (1920), TSingleHelper.Frac (1922), TSingleHelper.Fraction (1920)

76.107.16 TSingleHelper.SpecialType

Synopsis: Return the type of the single-sized floating point value.

Declaration: `function SpecialType : TFloatSpecial`

Visibility: public

Description: `SpecialType` checks whether the single-sized floating point value equals one of several special values, and returns an enumerated value describing which value this is. See TFloatSpecial (1635) for an explanation of the possible values.

See also: TFloatSpecial (1635)

76.107.17 TSingleHelper.Bytes

Synopsis: Indexed access to the individual bytes of the floating point value.

Declaration: `Property Bytes[AIndex: Cardinal]: Byte`

Visibility: public

Access: Read,Write

Description: `Bytes` can be used to get or set the various bytes that make up the single-sized floating point value. The index runs from 0 to `Size-1`.

See also: TSingleHelper.Words (1921), TSingleHelper.Size (1919)

76.107.18 TSingleHelper.Words

Synopsis: Indexed access to the words that make up the floating point value.

Declaration: `Property Words[AIndex: Cardinal]: Word`

Visibility: public

Access: Read,Write

Description: `Words` can be used to get or set the various bytes that make up the single-sized floating point value. The index runs from 0 to `(Size-1) div 2`.

See also: TSingleHelper.Bytes (1921), TSingleHelper.Size (1919)

76.107.19 TSingleHelper.Sign

Synopsis: Sign of the floating point value.

Declaration: `Property Sign : Boolean`

Visibility: public

Access: Read,Write

Description: `Sign` returns `True` if the sign bit of the value is set (i.e. it is a negative value) or `False` if it is not set (i.e. it is a positive value).

See also: TSingleHelper.Bytes (1921), TSingleHelper.Exp (1922), TSingleHelper.Frac (1922), TSingleHelper.Mantissa (1921), TSingleHelper.Fraction (1920), TSingleHelper.Exponent (1920)

76.107.20 TSingleHelper.Exp

Synopsis: The bit pattern of the exponent as stored in memory.

Declaration: `Property Exp : QWord`

Visibility: public

Access: Read,Write

Description: `Exp` is the internal representation of the Exponent (1920).

See also: TSingleHelper.Bytes (1921), TSingleHelper.Sign (1922), TSingleHelper.Frac (1922), TSingleHelper.Mantissa (1921), TSingleHelper.Fraction (1920), TSingleHelper.Exponent (1920)

76.107.21 TSingleHelper.Frac

Synopsis: Bitpattern that makes up the fractional part.

Declaration: `Property Frac : QWord`

Visibility: public

Access: Read,Write

Description: `Frac` is the bit pattern representing the fractional part (significand) including the preceding 1 (the hidden bit).

See also: TSingleHelper.Bytes (1921), TSingleHelper.Sign (1922), TSingleHelper.Exp (1922), TSingleHelper.Mantissa (1921), TSingleHelper.Fraction (1920), TSingleHelper.Exponent (1920)

76.108 TSmallIntHelper

76.108.1 Description

`TSmallIntHelper` contains some auxiliary routines for a SmallInt-typed ordinal value. It consists mainly of conversion routines to and from other types.

See also: TStringHelper (1926), TShortIntHelper (1910), TByteHelper (1855), TWordHelper (1960), TCardinalHelper (1859), TIntegerHelper (1888), TInt64Helper (1884), TQWordHelper (1906), TNativeIntHelper (1898), TNativeUIntHelper (1902)

76.108.2 Method overview

Page	Method	Description
1926	ClearBit	Set bit to 0.
1923	Parse	Convert from a string.
1925	SetBit	Set bit to 1.
1923	Size	Size, in bytes, of the SmallInt value.
1926	TestBit	Check bit.
1924	ToBinString	Convert to a binary string representation.
1924	ToBoolean	Convert to a boolean value.
1925	ToDouble	Convert to a double-sized floating point value.
1925	ToExtended	Convert to an extended-sized floating point value.
1926	ToggleBit	Invert bit.
1924	ToHexString	Convert to a hexadecimal string representation.
1925	ToSingle	Convert to an single-sized floating point value.
1923	ToString	Convert the value to string.
1924	TryParse	Try to convert a string to a SmallInt, report success or failure.

76.108.3 TSmallIntHelper.Parse

Synopsis: Convert from a string.

Declaration:
```
class function Parse(const AString: string) : SmallInt;  Static
```

Visibility: public

Description: `Parse` will attempt to convert the string `AString` to a SmallInt value. It uses the StrToInt (1793) function to perform the conversion, so no localization is taken into account.

Errors: If the string does not contain a valid SmallInt value, an EConvertError (1830) exception is raised.

See also: TSmallIntHelper.ToString (1923), TSmallIntHelper.TryParse (1924), StrToInt (1793)

76.108.4 TSmallIntHelper.Size

Synopsis: Size, in bytes, of the SmallInt value.

Declaration:
```
class function Size : Integer;  Static
```

Visibility: public

Description: `Size` returns the size (in SmallInts) of the SmallInt value. This is equivalent to `SizeOf(SmallInt)`.

Errors: None.

See also: SizeOf (1574)

76.108.5 TSmallIntHelper.ToString

Synopsis: Convert the value to string.

Declaration:
```
class function ToString(const AValue: SmallInt) : string;  Overload
                              ;  Static
function ToString : string;  Overload
```

Visibility: public

Description: `ToString` will, in the class function variant of this method, convert `AValue` to a string representation. In the regular method overloaded version of `ToString`, the SmallInt value itself is used. The IntToStr (1758) function is used to do the conversion.

See also: TSmallIntHelper.Parse (1923), IntToStr (1758)

76.108.6 TSmallIntHelper.TryParse

Synopsis: Try to convert a string to a SmallInt, report success or failure.

Declaration:
```
class function TryParse(const AString: string; out AValue: SmallInt)
                                : Boolean;  Static
```

Visibility: public

Description: `TryParse` attempts to convert the string `AString` to a SmallInt, and reports the success of the attempt. If the attempt is successful, then `True` is returned, and the actual value of the SmallInt is returned in `AValue`.

It uses the val (1635) function to perform the conversion, so no localization is taken into account.

See also: TSmallIntHelper.Parse (1923), Val (1598)

76.108.7 TSmallIntHelper.ToBoolean

Synopsis: Convert to a boolean value.

Declaration:
```
function ToBoolean : Boolean
```

Visibility: public

Description: `ToBoolean` converts the SmallInt value to a boolean: it returns `True` if the value is nonzero, `False` if it is zero.

See also: TSmallIntHelper.ToSingle (1925), TSmallIntHelper.ToDouble (1925), TSmallIntHelper.ToExtended (1925), TSmallIntHelper.ToString (1923), TSmallIntHelper.ToHexString (1924)

76.108.8 TSmallIntHelper.ToBinString

Synopsis: Convert to a binary string representation.

Declaration:
```
function ToBinString : string
```

Visibility: public

Description: `ToBinString` converts the smallint value to a binary string representation, showing all 16 bits.

See also: TSmallIntHelper.ToHexString (1924), TSmallIntHelper.ToString (1923)

76.108.9 TSmallIntHelper.ToHexString

Synopsis: Convert to a hexadecimal string representation.

Declaration:
```
function ToHexString : string;  Overload
function ToHexString(const AMinDigits: Integer) : string;  Overload
```

Visibility: public

Description: `ToHexString` converts the SmallInt value to a hexadecimal string representation. The `AMinDigits` argument specifies the minimal number of characters in the resulting string. The string will be left-padded with zeroes if the representation contains less than `AMinDigits` characters.

See also: TSmallIntHelper.ToBoolean (1924), TSmallIntHelper.ToSingle (1925), TSmallIntHelper.ToDouble (1925), TSmallIntHelper.ToString (1923), TSmallIntHelper.ToExtended (1925)

76.108.10 TSmallIntHelper.ToSingle

Synopsis: Convert to an single-sized floating point value.

Declaration: `function ToSingle : Single`

Visibility: public

Description: `ToSingle` converts the SmallInt value to a single-sized floating point value.

See also: TSmallIntHelper.ToBoolean (1924), TSmallIntHelper.ToDouble (1925), TSmallIntHelper.ToExtended (1925), TSmallIntHelper.ToString (1923), TSmallIntHelper.ToHexString (1924)

76.108.11 TSmallIntHelper.ToDouble

Synopsis: Convert to a double-sized floating point value.

Declaration: `function ToDouble : Double`

Visibility: public

Description: `ToDouble` converts the SmallInt value to a double-sized floating point value

See also: TSmallIntHelper.ToBoolean (1924), TSmallIntHelper.ToExtended (1925), TSmallIntHelper.ToSingle (1925), TSmallIntHelper.ToString (1923), TSmallIntHelper.ToHexString (1924)

76.108.12 TSmallIntHelper.ToExtended

Synopsis: Convert to an extended-sized floating point value.

Declaration: `function ToExtended : Extended`

Visibility: public

Description: `ToDouble` converts the SmallInt value to an extended-sized floating point value.

See also: TSmallIntHelper.ToBoolean (1924), TSmallIntHelper.ToSingle (1925), TSmallIntHelper.ToDouble (1925), TSmallIntHelper.ToString (1923), TSmallIntHelper.ToHexString (1924)

76.108.13 TSmallIntHelper.SetBit

Synopsis: Set bit to 1.

Declaration: `function SetBit(const Index: TSmallIntBitIndex) : SmallInt`

Visibility: public

Description: `SetBit` puts value 1 to bit number determined by argument `Index`.

See also: TSmallIntHelper.ClearBit (1926), TSmallIntHelper.ToggleBit (1926), TSmallIntHelper.TestBit (1926), TSmallIntHelper.HighestSetBitPos (1922), TSmallIntHelper.LowestSetBitPos (1922), TSmallIntHelper.SetBitsCount (1922), TSmallIntHelper.Bits (1922)

76.108.14 TSmallIntHelper.ClearBit

Synopsis: Set bit to 0.

Declaration: `function ClearBit(const Index: TSmallIntBitIndex) : SmallInt`

Visibility: public

Description: `ClearBit` puts value 0 to bit number determined by argument `Index`.

See also: TSmallIntHelper.SetBit (1925), TSmallIntHelper.ToggleBit (1926), TSmallIntHelper.TestBit (1926), TSmallIntHelper.HighestSetBitPos (1922), TSmallIntHelper.LowestSetBitPos (1922), TSmallIntHelper.SetBitsCount (1922), TSmallIntHelper.Bits (1922)

76.108.15 TSmallIntHelper.ToggleBit

Synopsis: Invert bit.

Declaration: `function ToggleBit(const Index: TSmallIntBitIndex) : SmallInt`

Visibility: public

Description: `ToggleBit` reads bit value from bit number determined by argument `Index`, inverts it (if it was 0 it becomes 1, and vice versa), and stores it back.

See also: TSmallIntHelper.SetBit (1925), TSmallIntHelper.ToggleBit (1926), TSmallIntHelper.TestBit (1926), TSmallIntHelper.HighestSetBitPos (1922), TSmallIntHelper.LowestSetBitPos (1922), TSmallIntHelper.SetBitsCount (1922), TSmallIntHelper.Bits (1922)

76.108.16 TSmallIntHelper.TestBit

Synopsis: Check bit.

Declaration: `function TestBit(const Index: TSmallIntBitIndex) : Boolean`

Visibility: public

Description: `TestBit` reads boolean value (true if bit is 1, and false if bit is 0) from bit number determined by argument `Index`.

See also: TSmallIntHelper.SetBit (1925), TSmallIntHelper.ToggleBit (1926), TSmallIntHelper.TestBit (1926), TSmallIntHelper.HighestSetBitPos (1922), TSmallIntHelper.LowestSetBitPos (1922), TSmallIntHelper.SetBitsCount (1922), TSmallIntHelper.Bits (1922)

76.109 TStringHelper

76.109.1 Description

`TStringHelper` adds various helper routines to the string type. These are mostly conversion routines, and some formatting routines.

For similarity to C-like languages, all the indexes in these helper routines are zero based.

See also: TGUIDHelper (1882)

76.109.2 Method overview

Page	Method	Description
1929	Compare	Compare 2 strings.
1930	CompareOrdinal	Compare 2 strings byte for byte.
1930	CompareText	Compare 2 strings case insensitively.
1936	CompareTo	Compare string to another.
1936	Contains	Check is the string contains another.
1930	Copy	Return a unique copy of a string.
1936	CopyTo	Copy part of the string to an array of characters.
1936	CountChar	Count the occurrences of a character.
1931	Create	Create a new string.
1937	DeQuotedString	Return a dequoted version of the string.
1931	EndsText	Check if one string is the ending of another.
1937	EndsWith	Check if the string is ended by another.
1931	Equals	Check if 2 strings are equal.
1932	Format	Format a string using provided arguments.
1937	GetHashCode	Get a hash code for the string.
1937	IndexOf	Find the position (index) of a string or character.
1938	IndexOfAny	Find the position (index) of any string or character in a list.
1939	IndexOfAnyUnquoted	Find the position (index) of any string or character in a list.
1938	IndexOfUnQuoted	Index of string, skipping quoted parts.
1939	Insert	Insert a string at a given position.
1940	IsDelimiter	Check whether a character at a given position is a delimiter.
1940	IsEmpty	Check whether the string is empty.
1932	IsNullOrEmpty	Check if a string is empty.
1932	IsNullOrWhiteSpace	Check if a string is empty or contains only whitespace characters.
1932	Join	Join a series of strings, separated using a given separator.
1940	LastDelimiter	Return the last position of one of a series of delimiters.
1940	LastIndexOf	Find the last position (index) of a string or character.
1941	LastIndexOfAny	Find the last position (index) of any string or character in a list.
1933	LowerCase	Return the lowercase version of a string.
1941	PadLeft	Pad the string on the left with an indicated character.
1942	PadRight	Pad the string on the right with an indicated character.
1933	Parse	Return a string representation of the argument.
1942	QuotedString	Return a quoted version of the string.
1942	Remove	Remove a number of characters from the string.
1943	Replace	Replace occurrences of one string with another.
1943	Split	Split a string in a number of parts.
1944	StartsWith	Check if one string starts with another.
1944	Substring	Return a part of the string.
1933	ToBoolean	Convert string to boolean.
1945	ToCharArray	Return the string as an array of characters.
1934	ToDouble	Convert string to double-sized floating point value.
1934	ToExtended	Convert string to extended-sized floating point value.
1934	ToInt64	Convert string to 64-bit signed integer.
1935	ToInteger	Convert string to 32-bit signed integer.
1945	ToLower	Convert to lowercase.
1945	ToLowerInvariant	Convert to lowercase.
1935	ToSingle	Convert string to single-sized floating point value.
1946	ToUpper	Convert to uppercase.
1946	ToUpperInvariant	Convert to uppercase.
1946	Trim	Strips a set of trim characters from the beginning and end of the string.
1947	TrimEnd	Alias for `TrimRight`.
1946	TrimLeft	Strips a set of trim characters from the beginning of the string.
1947	TrimRight	Strips a set of trim characters from the end of the string.
1947	TrimStart	Alias for `TrimLeft`.
1935	UpperCase	Return uppercase version of a string.

76.109.3 Property overview

Page	Properties	Access	Description
1947	Chars	r	Zero-based Indexed access to the characters in the string.
1948	Length	r	Return the length of the string.

76.109.4 TStringHelper.Compare

Synopsis: Compare 2 strings.

Declaration:
```
class function Compare(const A: string; const B: string) : Integer
                       ;  Overload;  Static
class function Compare(const A: string; const B: string;
                       IgnoreCase: Boolean) : Integer;  Overload;  Static
class function Compare(const A: string; const B: string;
                       Options: TCompareOptions) : Integer;  Overload
                       ;  Static
class function Compare(const A: string; IndexA: SizeInt;
                       const B: string; IndexB: SizeInt; ALen: SizeInt)
                        : Integer;  Overload;  Static
class function Compare(const A: string; IndexA: SizeInt;
                       const B: string; IndexB: SizeInt; ALen: SizeInt;
                       IgnoreCase: Boolean) : Integer;  Overload;  Static
class function Compare(const A: string; IndexA: SizeInt;
                       const B: string; IndexB: SizeInt; ALen: SizeInt;
                       Options: TCompareOptions) : Integer;  Overload
                       ;  Static
```

Visibility: public

Description: `Compare` compares strings `A` and `B`. It returns the following result:

- 0 if the strings are equal
- a negative number if `A` < `B`
- a positive number if `A` > `B`

The comparison can be influenced by using the appropriate overloaded version of the function.

- If the `IndexA` and `IndexB` parameters are present, the comparison starts at character index `IndexA` and `IndexB`. The indexes are zero-based.
- If the `ALen` parameters is present, then only the first `ALen` characters are compared. If not enough characters are present in either `A` or `B`, the comparison will include only as much characters as are present.
- If `IgnoreCase` is present and used, it determines whether the comparison is done case-sensitively. This form is deprecated, it is recommended to use the `AOptions` parameter and to include `coIgnoreCase`.
- The `AOptions` argument can be used to specify additional options. See TCompareOption (1412) for a list of possible values in this set.

Errors: None.

See also: TStringHelper.CompareOrdinal (1930), TCompareOption (1412), TStringHelper.CompareTo (1936)

76.109.5 TStringHelper.CompareOrdinal

Synopsis: Compare 2 strings byte for byte.

Declaration:
```
class function CompareOrdinal(const A: string; const B: string)
                               : Integer;  Overload;  Static
class function CompareOrdinal(const A: string; IndexA: SizeInt;
                              const B: string; IndexB: SizeInt;
                              ALen: SizeInt) : Integer;  Overload
                              ;  Static
```

Visibility: public

Description: `CompareOrdinal` compares 2 strings `A` and `B` bytewise. It is faster than a regular compare, but offers less options.

- 0 if the strings are equal
- a negative number if `A` < `B`
- a positive number if `A` > `B`

Optionally, a zero-based starting index for the compare can be given for each of the strings: `IndexA` and `IndexB`. In this case a maximum amount of characters (`ALen`) to be compared must also be specified.

Errors: None.

See also: TStringHelper.Compare (1929), TStringHelper.CompareTo (1936)

76.109.6 TStringHelper.CompareText

Synopsis: Compare 2 strings case insensitively.

Declaration:
```
class function CompareText(const A: string; const B: string) : Integer
                           ;  Static
```

Visibility: public

Description: `CompareText` simply calls SysUtils (1690) with the 2 passed arguments.

Errors: None.

See also: SysUtils (1690), TStringHelper.CompareTo (1936)

76.109.7 TStringHelper.Copy

Synopsis: Return a unique copy of a string.

Declaration:
```
class function Copy(const Str: string) : string;  Static
```

Visibility: public

Description: `Copy` will copy the string on which it operates, and makes sure the result has reference count 1.

See also: UniqueString (1635)

76.109.8 TStringHelper.Create

Synopsis: Create a new string.

Declaration:
```
class function Create(AChar: Char; ACount: SizeInt) : string;  Overload
                      ;  Static
class function Create(const AValue: Array of Char) : string;  Overload
                      ;  Static
class function Create(const AValue: Array of Char; StartIndex: SizeInt;
                      ALen: SizeInt) : string;  Overload;  Static
```

Visibility: public

Description: `Create` returns a new string with an initial value based on the passed arguments:

- A character (`ACHar`) and a count (`ACount`, in which case a string is returned of the specified length, filled with `AChar`.
- An array of characters `AValue`. The returned string will have the same length as the array, and the characters will be copied from the elements in the array. Optionally, a starting index `StartIndex` (zero based) and length `ALen` can be specified, in which case the string will contain at most `ALen` characters, which will have been copied starting at index `StartIndex`.

Errors: None.

See also: StringOfChar (1635)

76.109.9 TStringHelper.EndsText

Synopsis: Check if one string is the ending of another.

Declaration:
```
class function EndsText(const ASubText: string; const AText: string)
                        : Boolean;  Static
```

Visibility: public

Description: `EndsText` returns `True` if `AText` ends on `SubText`, i.e. whether the last characters in `AText` are the ones found in `SubText`. The comparison is done case insensitively.

See also: CompareText (1690), TStringHelper.EndsWith (1937)

76.109.10 TStringHelper.Equals

Synopsis: Check if 2 strings are equal.

Declaration:
```
class function Equals(const a: string; const b: string) : Boolean
                      ;  Overload;  Static
function Equals(const AValue: string) : Boolean;  Overload
```

Visibility: public

Description: `Equals` returns `True` if `A=B`, false otherwise.

See also: CompareText (1690), TStringHelper.Compare (1929)

76.109.11 TStringHelper.Format

Synopsis: Format a string using provided arguments.

Declaration:
```
class function Format(const AFormat: string; const args: Array of const)
                        : string;  Overload;  Static
function Format(const args: Array of const) : string;  Overload
```

Visibility: public

Description: `Format` just calls Sysutils.Format (1735), passing on `AFormat` and `Args`. If `AFormat` is omitted, the string itself is used as the formatting string.

See also: Sysutils.Format (1735)

76.109.12 TStringHelper.IsNullOrEmpty

Synopsis: Check if a string is empty.

Declaration:
```
class function IsNullOrEmpty(const AValue: string) : Boolean;  Static
```

Visibility: public

Description: `IsNullOrEmpty` returns `True` if string `A` has length 0.

See also: System.Length (1635), TStringHelper.IsNullOrWhiteSpace (1932)

76.109.13 TStringHelper.IsNullOrWhiteSpace

Synopsis: Check if a string is empty or contains only whitespace characters.

Declaration:
```
class function IsNullOrWhiteSpace(const AValue: string) : Boolean
                                    ;  Static
```

Visibility: public

Description: `IsNullOrEmpty` returns `True` if string `A` has length 0 or contains only whitespace characters (characters with ASCII code 32 or less).

See also: TStringHelper.IsNullOrEmpty (1932), Trim (1803)

76.109.14 TStringHelper.Join

Synopsis: Join a series of strings, separated using a given separator.

Declaration:
```
class function Join(const Separator: string;
                    const Values: Array of const) : string;  Overload
                    ;  Static
class function Join(const Separator: string;
                    const Values: Array of string) : string;  Overload
                    ;  Static
class function Join(const Separator: string;
                    const Values: Array of string; StartIndex: SizeInt;
                    ACount: SizeInt) : string;  Overload;  Static
```

Visibility: public

Description: `Join` will return a string that consists of all the strings in `Values` concatenated, with `Separator` between them. `Values` can be an array of strings, but can also be an array of arbitrary values: the values will first be transformed to a string representation.

In the case where an array of strings is passed in, a start index `StartIndex` and element count `ACount` can be passed in as well. Only the elements in the indicated range will then be used for the output.

Errors: Classes, pointers or interfaces will not be transformed, but will result in a 'unknown type: NNN' string being inserted.

See also: TStringHelper.Split (1943)

76.109.15 TStringHelper.LowerCase

Synopsis: Return the lowercase version of a string.

Declaration:
```
class function LowerCase(const S: string) : string;  Overload;  Static
```

Visibility: public

Description: `LowerCase` returns the lowercase version of the string `S`. This function is an alias for the SysUtils.LowerCase (1763) function.

See also: sysutils.LowerCase (1763)

76.109.16 TStringHelper.Parse

Synopsis: Return a string representation of the argument.

Declaration:
```
class function Parse(const AValue: Boolean) : string;  Overload;  Static
class function Parse(const AValue: Extended) : string;  Overload
                    ;  Static
class function Parse(const AValue: Int64) : string;  Overload;  Static
class function Parse(const AValue: Integer) : string;  Overload;  Static
```

Visibility: public

Description: `Parse` will return a string representation of the argument `AValue`. How this is done depends on the type of the `AValue`argument:

BooleanThe result is formatted using BoolToStr (1685).

ExtendedThe result is formatted using FloatToStr (1729).

Int64 The result is formatted using IntToStr (1758).

Integerhe result is formatted using IntToStr (1758).

See also: BoolToStr (1685), FloatToStr (1729), IntToStr (1758)

76.109.17 TStringHelper.ToBoolean

Synopsis: Convert string to boolean.

Declaration:
```
class function ToBoolean(const S: string) : Boolean;  Overload;  Static
function ToBoolean : Boolean;  Overload
```

Visibility: public

Description: `ToBoolean` returns the contents of the string `S` as a boolean (if possible). It uses StrToBool (1787) to convert the value to a boolean.

Errors: If the string contains a value that cannot be translated to a boolean, an EConvertError (1830) exception may be raised.

See also: StrToBool (1787), EConvertError (1830), TStringHelper.ToDouble (1934), TStringHelper.ToExtended (1934), TStringHelper.ToInt64 (1934), TStringHelper.ToInteger (1935), TStringHelper.ToSingle (1935)

76.109.18 TStringHelper.ToDouble

Synopsis: Convert string to double-sized floating point value.

Declaration:
```
class function ToDouble(const S: string) : Double;  Overload;  Static
function ToDouble : Double;  Overload
```

Visibility: public

Description: `ToDouble` returns the contents of the string `S` as a Double-sized floating point value (if possible). It uses StrToFloat (1791) to convert the value to a floating point value.

Errors: If the string contains a value that cannot be translated to a floating point value, an EConvertError (1830) exception may be raised.

See also: StrToFloat (1791), EConvertError (1830), TStringHelper.ToBoolean (1933), TStringHelper.ToExtended (1934), TStringHelper.ToInt64 (1934), TStringHelper.ToInteger (1935), TStringHelper.ToSingle (1935)

76.109.19 TStringHelper.ToExtended

Synopsis: Convert string to extended-sized floating point value.

Declaration:
```
class function ToExtended(const S: string) : Extended;  Overload
                           ;  Static
function ToExtended : Extended;  Overload
```

Visibility: public

Description: `ToExtended` returns the contents of the string `S` as a Extended-sized floating point value (if possible). It uses StrToFloat (1791) to convert the value to a floating point value.

Errors: If the string contains a value that cannot be translated to a floating point value, an EConvertError (1830) exception may be raised.

See also: StrToFloat (1791), EConvertError (1830), TStringHelper.ToBoolean (1933), TStringHelper.ToDouble (1934), TStringHelper.ToInt64 (1934), TStringHelper.ToInteger (1935), TStringHelper.ToSingle (1935)

76.109.20 TStringHelper.ToInt64

Synopsis: Convert string to 64-bit signed integer.

Declaration:
```
class function ToInt64(const S: string) : Int64;  Overload;  Static
function ToInt64 : Int64;  Overload
```

Visibility: public

Description: `ToInt64` returns the contents of the string `S` as a 64-bit signed integer value (if possible). It uses StrToInt64 (1794) to convert the value to an 64-bit sized integer value.

Errors: If the string contains a value that cannot be translated to an integer value, an EConvertError (1830) exception may be raised.

See also: StrToInt64 (1794), EConvertError (1830), TStringHelper.ToBoolean (1933), TStringHelper.ToDouble (1934), TStringHelper.ToExtended (1934), TStringHelper.ToInteger (1935), TStringHelper.ToSingle (1935)

76.109.21 TStringHelper.ToInteger

Synopsis: Convert string to 32-bit signed integer.

Declaration: `class function ToInteger(const S: string) : Integer; Overload; Static function ToInteger : Integer; Overload`

Visibility: public

Description: `ToInteger` returns the contents of the string `S` as a 32-bit signed integer value (if possible). It uses StrToInt64 (1794) to convert the value to an 64-bit sized integer value.

Errors: If the string contains a value that cannot be translated to an integer value, an EConvertError (1830) exception may be raised.

See also: StrToInt64 (1794), EConvertError (1830), TStringHelper.ToBoolean (1933), TStringHelper.ToDouble (1934), TStringHelper.ToExtended (1934), TStringHelper.ToInt64 (1934), TStringHelper.ToSingle (1935)

76.109.22 TStringHelper.ToSingle

Synopsis: Convert string to single-sized floating point value.

Declaration: `class function ToSingle(const S: string) : Single; Overload; Static function ToSingle : Single; Overload`

Visibility: public

Description: `ToSingle` returns the contents of the string `S` as a single-sized floating point value (if possible). It uses StrToFloat (1791) to convert the value to a floating point value.

Errors: If the string contains a value that cannot be translated to a floating point value, an EConvertError (1830) exception may be raised.

See also: StrToFloat (1791), EConvertError (1830), TStringHelper.ToBoolean (1933), TStringHelper.ToDouble (1934), TStringHelper.ToInt64 (1934), TStringHelper.ToInteger (1935), TStringHelper.ToExtended (1934)

76.109.23 TStringHelper.UpperCase

Synopsis: Return uppercase version of a string.

Declaration: `class function UpperCase(const S: string) : string; Overload; Static`

Visibility: public

Description: `UpperCase` returns an uppercase version of the string `S`.

See also: #rtl.sysutils.UpperCase (1814)

76.109.24 TStringHelper.CompareTo

Synopsis: Compare string to another.

Declaration:
```
function CompareTo(const B: string) : Integer
```

Visibility: public

Description: `CompareTo` will compare the string value to the string `B`. It returns the following result:

- 0 if the strings are equal
- a negative number if `Self` < `B`
- a positive number if `Self` > `B`

The StrComp (1774) function is used for this.

See also: TStringHelper.Compare (1929), TStringHelper.CompareOrdinal (1930), StrComp (1774)

76.109.25 TStringHelper.Contains

Synopsis: Check is the string contains another.

Declaration:
```
function Contains(const AValue: string) : Boolean
```

Visibility: public

Description: `Contains` returns `True` if the string value contains `AValue`, i.e. it returns `Pos(Self,AValue)>0`.

See also: Pos (1543)

76.109.26 TStringHelper.CopyTo

Synopsis: Copy part of the string to an array of characters.

Declaration:
```
procedure CopyTo(SourceIndex: SizeInt; var destination: Array of Char;
                 DestinationIndex: SizeInt; ACount: SizeInt)
```

Visibility: public

Description: `CopyTo` copies at most `ACount` characters from the source string (`Self`) to the array `Destination`, starting at (zero based) index `SourceIndex`. The characters are copied to the array starting at position `DestinationIndex` (zero based).

See also: Copy (1473)

76.109.27 TStringHelper.CountChar

Synopsis: Count the occurrences of a character.

Declaration:
```
function CountChar(const C: Char) : SizeInt
```

Visibility: public

Description: `CountChar` returns the number of occurrences of `C` in the string (`Self`).

See also: TStringHelper.IndexOf (1937), TStringHelper.IndexOfAny (1938)

76.109.28 TStringHelper.DeQuotedString

Synopsis: Return a dequoted version of the string.

Declaration:
```
function DeQuotedString : string;  Overload
function DeQuotedString(const AQuoteChar: Char) : string;  Overload
```

Visibility: public

Description: `DeQuotedString` will return a dequoted version of the string (`Self`), where the quote character is `AQuoteChar` (default: a single quote '). The string must start and end with the quote character, or it is returned as-is. Any double occurrences of the quote character `AQuoteChar` will be returned as a single quote.

See also: AnsiQuotedString (1635), AnsiExtractQuotedString (1635)

76.109.29 TStringHelper.EndsWith

Synopsis: Check if the string is ended by another.

Declaration:
```
function EndsWith(const AValue: string) : Boolean;  Overload
function EndsWith(const AValue: string; IgnoreCase: Boolean) : Boolean
                 ;  Overload
```

Visibility: public

Description: `EndsWith` returns `True` if the string (`Self`) ends on `AValue`, i.e. whether the last characters in `Self` are the ones found in `AValue`. The comparison is done case insensitively depending on the `IgnoreCase` argument (which is false by default).

See also: CompareText (1690), TStringHelper.EndsText (1931)

76.109.30 TStringHelper.GetHashCode

Synopsis: Get a hash code for the string.

Declaration:
```
function GetHashCode : Integer
```

Visibility: public

Description: `GetHashCode` returns a hash value for the string (`Self`).

76.109.31 TStringHelper.IndexOf

Synopsis: Find the position (index) of a string or character.

Declaration:
```
function IndexOf(AValue: Char) : SizeInt;  Overload
function IndexOf(const AValue: string) : SizeInt;  Overload
function IndexOf(AValue: Char; StartIndex: SizeInt) : SizeInt;  Overload
function IndexOf(const AValue: string; StartIndex: SizeInt) : SizeInt
                ;  Overload
function IndexOf(AValue: Char; StartIndex: SizeInt; ACount: SizeInt)
                 : SizeInt;  Overload
function IndexOf(const AValue: string; StartIndex: SizeInt;
                ACount: SizeInt) : SizeInt;  Overload
```

Visibility: public

Description: `IndexOf` returns the zero-based index of `AValue` in the string (`Self`). The value to search for can be a character or string (`AValue`).

if no match is found, -1 is returned.

The search can be refined by specifying a (zero based) index `StartIndex`. When specified, the search will start at the given character. Not specifying this option is equivalent to specifying zero.

If `ACount` is given, at most `ACount` characters in the source string will be considered for the search. This count includes the starting character. Not specifying this argument is equivalent to specifying the amount of remaining characters.

See also: Pos (1543), TStringHelper.IndexOfUnQuoted (1938), TStringHelper.IndexOfAny (1938), TStringHelper.IndexOfAnyUnquote (1939), TStringHelper.LastIndexOf (1940), TStringHelper.LastIndexOfAny (1941)

76.109.32 TStringHelper.IndexOfUnQuoted

Synopsis: Index of string, skipping quoted parts.

Declaration:
```
function IndexOfUnQuoted(const AValue: string; StartQuote: Char;
                         EndQuote: Char; StartIndex: SizeInt) : SizeInt
                         ;  Overload
```

Visibility: public

Description: `IndexOfUnQuoted` will return the zero-based index of `AValue` in the string (`Self`), but disregards any quoted parts in the string. A quoted part is determined by a starting `StartQuote` and end-quote character `EndQuote`. The search starts at (zero-based) position `StartIndex`, which is by default 0.

See also: Pos (1543), TStringHelper.IndexOf (1937), TStringHelper.IndexOfAny (1938), TStringHelper.IndexOfAnyUnquoted (1939), TStringHelper.LastIndexOf (1940), TStringHelper.LastIndexOfAny (1941)

76.109.33 TStringHelper.IndexOfAny

Synopsis: Find the position (index) of any string or character in a list.

Declaration:
```
function IndexOfAny(const AnyOf: Array of Char) : SizeInt;  Overload
function IndexOfAny(const AnyOf: Array of Char; StartIndex: SizeInt)
                    : SizeInt;  Overload
function IndexOfAny(const AnyOf: Array of Char; StartIndex: SizeInt;
                    ACount: SizeInt) : SizeInt;  Overload
function IndexOfAny(const AnyOf: Array of string) : SizeInt;  Overload
function IndexOfAny(const AnyOf: Array of string; StartIndex: SizeInt)
                    : SizeInt;  Overload
function IndexOfAny(const AnyOf: Array of string; StartIndex: SizeInt;
                    ACount: SizeInt) : SizeInt;  Overload
function IndexOfAny(const AnyOf: Array of string; StartIndex: SizeInt;
                    ACount: SizeInt; out AMatch: SizeInt) : SizeInt
                    ;  Overload
```

Visibility: public

Description: `IndexOfAny` returns the zero-based index of the first matching element in an array of characters or strings (`AnyOf`).

if no match is found, -1 is returned.

The search can be refined by specifying a (zero-based) index `StartIndex`. When specified, the search will start at the given character. Not specifying this option is equivalent to specifying zero.

If `ACount` is given, at most `ACount` characters in the source string will be considered for the search. This count includes the starting character. Not specifying this argument is equivalent to specifying the amount of remaining characters.

If `AMatch` is specified, on successful return it will contain the index of the element in the array that contains the found match.

See also: Pos (1543), TStringHelper.IndexOf (1937), TStringHelper.IndexOfUnquoted (1938), TStringHelper.IndexOfAnyUnquoted (1939), TStringHelper.LastIndexOf (1940), TStringHelper.LastIndexOfAny (1941)

76.109.34 TStringHelper.IndexOfAnyUnquoted

Synopsis: Find the position (index) of any string or character in a list.

Declaration:
```
function IndexOfAnyUnquoted(const AnyOf: Array of Char;
                            StartQuote: Char; EndQuote: Char) : SizeInt
                            ;  Overload
function IndexOfAnyUnquoted(const AnyOf: Array of Char;
                            StartQuote: Char; EndQuote: Char;
                            StartIndex: SizeInt) : SizeInt;  Overload
function IndexOfAnyUnquoted(const AnyOf: Array of Char;
                            StartQuote: Char; EndQuote: Char;
                            StartIndex: SizeInt; ACount: SizeInt)
                             : SizeInt;  Overload
function IndexOfAnyUnquoted(const AnyOf: Array of string;
                            StartQuote: Char; EndQuote: Char;
                            StartIndex: SizeInt; out Matched: SizeInt)
                             : SizeInt;  Overload
```

Visibility: public

Description: `IndexOfAnyUnQuoted` returns the zero-based index of the first matching element in an array of characters or strings (`AnyOf`), but disregards any quoted parts in the string. A quoted part is determined by a starting `StartQuote` and end-quote character `EndQuote`.

if no match is found, -1 is returned.

The search can be refined by specifying a (zero-based) index `StartIndex`. When specified, the search will start at the given character. Not specifying this option is equivalent to specifying zero.

If `ACount` is given, at most `ACount` characters in the source string will be considered for the search. This count includes the starting character. Not specifying this argument is equivalent to specifying the amount of remaining characters.

If `AMatch` is specified, on successful return it will contain the index of the element in the array that contains the found match.

See also: Pos (1543), TStringHelper.IndexOf (1937), TStringHelper.IndexOfUnquoted (1938), TStringHelper.IndexOfAny (1938), TStringHelper.LastIndexOf (1940), TStringHelper.LastIndexOfAny (1941)

76.109.35 TStringHelper.Insert

Synopsis: Insert a string at a given position.

Declaration:
```
function Insert(StartIndex: SizeInt; const AValue: string) : string
```

Visibility: public

Description: `Insert` inserts the string `AValue` in a string (`Self`) at the (zero-based) position `StartIndex`. It returns the resulting string (`Self`). If `StartIndex` is less than zero, it is set to zero. If `StartIndex` is higher than the length, `AValue` is appended to the string.

See also: Insert (1521)

76.109.36 TStringHelper.IsDelimiter

Synopsis: Check whether a character at a given position is a delimiter.

Declaration:
```
function IsDelimiter(const Delimiters: string; Index: SizeInt) : Boolean
```

Visibility: public

Description: `IsDelimiter` returns `True` if the character at (zero-based) position `Index` is one of the delimiter characters in `Delimiters`. If `Index` is out of range, `False` is returned.

See also: IsDelimiter (1759)

76.109.37 TStringHelper.IsEmpty

Synopsis: Check whether the string is empty.

Declaration:
```
function IsEmpty : Boolean
```

Visibility: public

Description: `IsEmpty` returns `True` if the string has length zero.

See also: TStringHelper.IsNullOrEmpty (1932), TStringHelper.Length (1948), Length (1528)

76.109.38 TStringHelper.LastDelimiter

Synopsis: Return the last position of one of a series of delimiters.

Declaration:
```
function LastDelimiter(const Delims: string) : SizeInt
```

Visibility: public

Description: `LastDelimiter` searches the string backwards for an occurrence of one of the characters in `Delims`, and returns the corresponding (zero-based) index in the string. If no occurrence is found, -1 is returned.

See also: TStringHelper.isDelimiter (1940), TStringHelper.IndexOf (1937), TStringHelper.IndexOfAny (1938), TStringHelper.LastIndexOf (1940), TStringHelper.LastIndexOfAny (1941)

76.109.39 TStringHelper.LastIndexOf

Synopsis: Find the last position (index) of a string or character.

Declaration:
```
function LastIndexOf(AValue: Char) : SizeInt;  Overload
function LastIndexOf(const AValue: string) : SizeInt;  Overload
function LastIndexOf(AValue: Char; AStartIndex: SizeInt) : SizeInt
                     ;  Overload
function LastIndexOf(const AValue: string; AStartIndex: SizeInt)
```

```
                        : SizeInt;  Overload
function LastIndexOf(AValue: Char; AStartIndex: SizeInt;
                     ACount: SizeInt) : SizeInt;  Overload
function LastIndexOf(const AValue: string; AStartIndex: SizeInt;
                     ACount: SizeInt) : SizeInt;  Overload
```

Visibility: public

Description: `LasyIndexOf` returns the zero-based index of the last occurrence of `AValue` in the string (`Self`). The value to search for can be a character or string (`AValue`).

if no match is found, -1 is returned.

The search can be refined by specifying a (zero based) index `StartIndex`. When specified, the search will start at the given character and proceeds towards the beginning of the string. Not specifying this option is equivalent to specifying `length-1`.

If `ACount` is given, at most `ACount` characters in the source string will be considered for the search. This count includes the starting character. Not specifying this argument is equivalent to specifying `StartIndex+1`.

See also: Pos (1543), TStringHelper.IndexOfUnQuoted (1938), TStringHelper.IndexOfAny (1938), TStringHelper.IndexOfAnyUnquote (1939), TStringHelper.IndexOf (1937), TStringHelper.LastIndexOfAny (1941)

76.109.40 TStringHelper.LastIndexOfAny

Synopsis: Find the last position (index) of any string or character in a list.

Declaration:
```
function LastIndexOfAny(const AnyOf: Array of Char) : SizeInt;  Overload
function LastIndexOfAny(const AnyOf: Array of Char;
                        AStartIndex: SizeInt) : SizeInt;  Overload
function LastIndexOfAny(const AnyOf: Array of Char;
                        AStartIndex: SizeInt; ACount: SizeInt) : SizeInt
                        ;  Overload
```

Visibility: public

Description: `LastIndexOfAny` returns the zero-based index of the first matching element in an array of characters or strings (`AnyOf`).

if no match is found, -1 is returned.

The search can be refined by specifying a (zero-based) index `StartIndex`. When specified, the search will start at the given character and proceeds to the start of the string. Not specifying this option is equivalent to specifying `Length-1`.

If `ACount` is given, at most `ACount` characters in the source string will be considered for the search. This count includes the starting character. Not specifying this argument is equivalent to specifying `StartIndex+1`.

See also: Pos (1543), TStringHelper.IndexOf (1937), TStringHelper.IndexOfUnquoted (1938), TStringHelper.IndexOfAnyUnquoted (1939), TStringHelper.LastIndexOf (1940), TStringHelper.IndexOfAny (1938)

76.109.41 TStringHelper.PadLeft

Synopsis: Pad the string on the left with an indicated character.

Declaration:
```
function PadLeft(ATotalWidth: SizeInt) : string;  Overload
function PadLeft(ATotalWidth: SizeInt; PaddingChar: Char) : string
                 ;  Overload
```

Visibility: public

Description: `PadLeft` pads the string (`Self`) on the left (i.e. at the beginning) till it reaches length `ATotalWidth` with character `PaddingChar`. If `PaddingChar` is omitted, a space is used.

See also: TStringHelper.PadRight (1942)

76.109.42 TStringHelper.PadRight

Synopsis: Pad the string on the right with an indicated character.

Declaration:
```
function PadRight(ATotalWidth: SizeInt) : string;  Overload
function PadRight(ATotalWidth: SizeInt; PaddingChar: Char) : string
                 ;  Overload
```

Visibility: public

Description: `PadLeft` pads the string (`Self`) on the right (i.e. at the end) till it reaches length `ATotalWidth` with character `PaddingChar`. If `PaddingChar` is omitted, a space is used.

See also: TStringHelper.PadLeft (1941)

76.109.43 TStringHelper.QuotedString

Synopsis: Return a quoted version of the string.

Declaration:
```
function QuotedString : string;  Overload
function QuotedString(const AQuoteChar: Char) : string;  Overload
```

Visibility: public

Description: `QuotedString` returns a quoted version of the string (`Self`). The quote character is specified in `AQuoteChar`, which by default is the double quote ("). Any existing quote characters occurrences will be doubled.

See also: QuotedStr (1765), AnsiQuotedStr (1675)

76.109.44 TStringHelper.Remove

Synopsis: Remove a number of characters from the string.

Declaration:
```
function Remove(StartIndex: SizeInt) : string;  Overload
function Remove(StartIndex: SizeInt; ACount: SizeInt) : string
               ;  Overload
```

Visibility: public

Description: `Remove` removes `ACount` characters from the string, starting at (zero-based) index `StartIndex`. If `ACount` is omitted, all remaining characters are removed.

See also: Delete (1477), TStringHelper.Replace (1943), TStringHelper.Insert (1939)

76.109.45 TStringHelper.Replace

Synopsis: Replace occurrences of one string with another.

Declaration:
```
function Replace(OldChar: Char; NewChar: Char) : string;  Overload
function Replace(OldChar: Char; NewChar: Char;
                 ReplaceFlags: TReplaceFlags) : string;  Overload
function Replace(const OldValue: string; const NewValue: string)
                  : string;  Overload
function Replace(const OldValue: string; const NewValue: string;
                 ReplaceFlags: TReplaceFlags) : string;  Overload
```

Visibility: public

Description: `Replace` will replace all occurrences of `OldChar` with `NewChar` or `OldValue` with `NewValue`. The search is case sensitive.

The behaviour of the `Replace` call can be controlled with the optional `ReplaceFlags` parameter:

If `rfReplaceAll` is in the `ReplaceFlags`, then all occurrences will be replaced, otherwise only the first occurrence is replaced.

If `rfCaseInsensitive` is in the `ReplaceFlags`, then the search for `OldChar` or `OldValue` is performed ignoring case.

Not specifying the `ReplaceFlags` parameter is therefore equivalent to specifying `[rfReplaceAll]`.

See also: StringReplace (1778), TReplaceFlags (1659)

76.109.46 TStringHelper.Split

Synopsis: Split a string in a number of parts.

Declaration:
```
function Split(const Separators: Array of Char) : TStringArray
               ;  Overload
function Split(const Separators: Array of Char; ACount: SizeInt)
                : TStringArray;  Overload
function Split(const Separators: Array of Char;
               Options: TStringSplitOptions) : TStringArray;  Overload
function Split(const Separators: Array of Char; ACount: SizeInt;
               Options: TStringSplitOptions) : TStringArray;  Overload
function Split(const Separators: Array of string) : TStringArray
               ;  Overload
function Split(const Separators: Array of string; ACount: SizeInt)
                : TStringArray;  Overload
function Split(const Separators: Array of string;
               Options: TStringSplitOptions) : TStringArray;  Overload
function Split(const Separators: Array of string; ACount: SizeInt;
               Options: TStringSplitOptions) : TStringArray;  Overload
function Split(const Separators: Array of Char; AQuote: Char)
                : TStringArray;  Overload
function Split(const Separators: Array of Char; AQuoteStart: Char;
               AQuoteEnd: Char) : TStringArray;  Overload
function Split(const Separators: Array of Char; AQuoteStart: Char;
               AQuoteEnd: Char; Options: TStringSplitOptions)
                : TStringArray;  Overload
function Split(const Separators: Array of Char; AQuoteStart: Char;
               AQuoteEnd: Char; ACount: SizeInt) : TStringArray
```

```
                ;  Overload
function Split(const Separators: Array of Char; AQuoteStart: Char;
                AQuoteEnd: Char; ACount: SizeInt;
                Options: TStringSplitOptions) : TStringArray;  Overload
function Split(const Separators: Array of string; AQuote: Char)
                 : TStringArray;  Overload
function Split(const Separators: Array of string; AQuoteStart: Char;
                AQuoteEnd: Char) : TStringArray;  Overload
function Split(const Separators: Array of string; AQuoteStart: Char;
                AQuoteEnd: Char; Options: TStringSplitOptions)
                 : TStringArray;  Overload
function Split(const Separators: Array of string; AQuoteStart: Char;
                AQuoteEnd: Char; ACount: SizeInt) : TStringArray
                ;  Overload
function Split(const Separators: Array of string; AQuoteStart: Char;
                AQuoteEnd: Char; ACount: SizeInt;
                Options: TStringSplitOptions) : TStringArray;  Overload
```

Visibility: public

Description: `Split` will split the string (`Self`) using `Separators` as separator characters.

If `ACount` is supplied, then at most `ACount` strings will be included in the result. The default behaviour is to supply all strings.

if `Options` contains `ExcludeEmpty` then no empty strings will be included in the result. Empty strings may be included if multiple successive separator characters are found in the source string, but not for the last character: If the last characters is a separator string, the 'empty string' behind it is not added to the result.

If `AQuoteStart` and `AQuoteEnd` are supplied, then no splitting will be performed between `AQuoteStart` and `AQuoteEnd` characters.

See also: TStringHelper.Join (1932), TStringHelper.IndexOfAny (1938), TStringHelper.IndexOfAnyUnquoted (1939)

76.109.47 TStringHelper.StartsWith

Synopsis: Check if one string starts with another.

Declaration:
```
function StartsWith(const AValue: string) : Boolean;  Overload
function StartsWith(const AValue: string; IgnoreCase: Boolean) : Boolean
                  ;  Overload
```

Visibility: public

Description: `StartsWith` will return `True` if the first characters of the string (`Self`) equal the string `AValue`. If `IgnoreCase` is `True`, then the comparison is done case insensitive. The default is to compare strings case sensitively.

See also: TStringHelper.EndsWith (1937)

76.109.48 TStringHelper.Substring

Synopsis: Return a part of the string.

Declaration:
```
function Substring(AStartIndex: SizeInt) : string;  Overload
function Substring(AStartIndex: SizeInt; ALen: SizeInt) : string
                   ;  Overload
```

Visibility: public

Description: SubString returns the portion of the string starting at (zero-based) index AStartIndex with length ALen. If ALen is omitted, then all remaining characters are returned.

See also: TStringHelper.Insert (1939), Copy (1473), TStringHelper.ToCharArray (1945)

76.109.49 TStringHelper.ToCharArray

Synopsis: Return the string as an array of characters.

Declaration:
```
function ToCharArray : TCharArray;  Overload
function ToCharArray(AStartIndex: SizeInt; ALen: SizeInt) : TCharArray
                     ;  Overload
```

Visibility: public

Description: ToCharArray returns part of the string (Self) as an array of characters. The characters are copied starting at (zero-based) index AStartIndex and at most ALen characters are copied. If AStartIndex and ALen are omitted, all the characters are returned.

Errors: If AStartIndex is less than zero, an EAccessViolation (1829) exception may be raised.

See also: TStringHelper.Substring (1944), TStringHelper.Insert (1939), Copy (1473)

76.109.50 TStringHelper.ToLower

Synopsis: Convert to lowercase.

Declaration:
```
function ToLower : string;  Overload
```

Visibility: public

Description: ToLower returns a lowercase version of the string.

See also: TStringHelper.LowerCase (1933), TStringHelper.ToUpper (1946), TStringHelper.ToUpperInvariant (1946)

76.109.51 TStringHelper.ToLowerInvariant

Synopsis: Convert to lowercase.

Declaration:
```
function ToLowerInvariant : string
```

Visibility: public

Description: ToLowerInvariant is equal to TStringHelper.ToLower (1945)

See also: TStringHelper.ToLower (1945), TStringHelper.ToUpper (1946), TStringHelper.ToUpperInvariant (1946)

76.109.52 TStringHelper.ToUpper

Synopsis: Convert to uppercase.

Declaration:
```
function ToUpper : string;  Overload
```

Visibility: public

Description: `>ToUpper` returns an uppercase version of the string.

See also: TStringHelper.ToLower (1945), TStringHelper.ToLowerInvariant (1945), TStringHelper.ToUpperInvariant (1946)

76.109.53 TStringHelper.ToUpperInvariant

Synopsis: Convert to uppercase.

Declaration:
```
function ToUpperInvariant : string
```

Visibility: public

Description: `ToUpperInvariant` is equal to TStringHelper.ToUpper (1946)

See also: TStringHelper.ToLower (1945), TStringHelper.ToLowerInvariant (1945), TStringHelper.ToUpper (1946)

76.109.54 TStringHelper.Trim

Synopsis: Strips a set of trim characters from the beginning and end of the string.

Declaration:
```
function Trim : string;  Overload
function Trim(const ATrimChars: Array of Char) : string;  Overload
```

Visibility: public

Description: `Trim` returns the string (`Self`), removing all characters occurring in `ATrimChars` from the start and end of the string. If `ATrimChars` is not specified, all characters with ASCII code 32 or lower are assumed.

See also: Trim (1803), TStringHelper.TrimLeft (1946), TStringHelper.TrimRight (1947)

76.109.55 TStringHelper.TrimLeft

Synopsis: Strips a set of trim characters from the beginning of the string.

Declaration:
```
function TrimLeft : string;  Overload
function TrimLeft(const ATrimChars: Array of Char) : string;  Overload
```

Visibility: public

Description: `Trim` returns the string (`Self`), removing all characters occurring in `ATrimChars` from the start of the string. If `ATrimChars` is not specified, all characters with ASCII code 32 or lower are assumed.

See also: TrimLeft (1803), TStringHelper.Trim (1946), TStringHelper.TrimRight (1947)

76.109.56 TStringHelper.TrimRight

Synopsis: Strips a set of trim characters from the end of the string.

Declaration:
```
function TrimRight : string;  Overload
function TrimRight(const ATrimChars: Array of Char) : string;  Overload
```

Visibility: public

Description: `Trim` returns the string (`Self`), removing all characters occurring in `ATrimChars` from the end of the string. If `ATrimChars` is not specified, all characters with ASCII code 32 or lower are assumed.

See also: Trim (1803), TStringHelper.Trim (1946), TStringHelper.TrimLeft (1946)

76.109.57 TStringHelper.TrimEnd

Synopsis: Alias for `TrimRight`.

Declaration:
```
function TrimEnd(const ATrimChars: Array of Char) : string
```

Visibility: public

Description: `TrimEnd` is an alias for TStringHelper.TrimRight (1947)

See also: TStringHelper.TrimRight (1947), TStringHelper.TrimStart (1947)

76.109.58 TStringHelper.TrimStart

Synopsis: Alias for `TrimLeft`.

Declaration:
```
function TrimStart(const ATrimChars: Array of Char) : string
```

Visibility: public

Description: `TrimStart` is an alias for TStringHelper.TrimLeft (1946)

See also: TStringHelper.TrimEnd (1947), TStringHelper.TrimLeft (1946)

76.109.59 TStringHelper.Chars

Synopsis: Zero-based Indexed access to the characters in the string.

Declaration:
```
Property Chars[AIndex: SizeInt]: Char
```

Visibility: public

Access: Read

Description: `Chars` provides zero-based indexed access to the characters in the string. The first character is at index 0, the last at `Length-1`.

See also: TStringHelper.Length (1948)

76.109.60 TStringHelper.Length

Synopsis: Return the length of the string.

Declaration: `Property &Length : SizeInt`

Visibility: public

Access: Read

Description: `Length` returns the length (in bytes for single-byte strings) of the string.

See also: Length (1528)

76.110 TUnicodeEncoding

76.110.1 Description

`TUnicodeEncoding` is the encoding class used to represent the UTF-16 encoding.

See also: TUnicodeEncoding (1948), TUTF7Encoding (1956), TMBCSEncoding (1894), TBigendianUnicodeEncoding (1851)

76.110.2 Method overview

Page	Method	Description
1948	Clone	Clone a `TUnicodeEncoding` instance.
1948	Create	Create a new instance of the `TUnicodeEncoding` class.
1949	GetMaxByteCount	Return max number of bytes needed to represent a string.
1949	GetMaxCharCount	Return max number of characters that can be represented by an array of bytes.
1949	GetPreamble	Return BOM marker bytes.

76.110.3 TUnicodeEncoding.Create

Synopsis: Create a new instance of the `TUnicodeEncoding` class.

Declaration: `constructor Create;  Virtual`

Visibility: public

Description: `Create` creates a new instance of the `TUnicodeEncoding` class and sets the codepage to `CP_UTF16`.

See also: TEncoding.CodePage (1873), TEncoding (1869)

76.110.4 TUnicodeEncoding.Clone

Synopsis: Clone a `TUnicodeEncoding` instance.

Declaration: `function Clone : TEncoding;  Override`

Visibility: public

Description: `CLone` overrides TEncoding.Clone (1870) to provide a clone of the `TUnicodeEncoding` instance.

See also: TEncoding.Clone (1870)

76.110.5 TUnicodeEncoding.GetMaxByteCount

Synopsis: Return max number of bytes needed to represent a string.

Declaration: `function GetMaxByteCount(CharCount: Integer) : Integer;  Override`

Visibility: public

Description: `GetMaxByteCount` overrides TEncoding.GetMaxByteCount (1872) to return the maximum number of bytes needed to represent a string.

See also: TEncoding.GetMaxByteCount (1872)

76.110.6 TUnicodeEncoding.GetMaxCharCount

Synopsis: Return max number of characters that can be represented by an array of bytes.

Declaration: `function GetMaxCharCount(ByteCount: Integer) : Integer;  Override`

Visibility: public

Description: `GetMaxCharCount` overrides TEncoding.GetMaxCharCount (1872) to return the maximum number of bytes needed to represent a string.

See also: TEncoding.GetMaxCharCount (1872)

76.110.7 TUnicodeEncoding.GetPreamble

Synopsis: Return BOM marker bytes.

Declaration: `function GetPreamble : TBytes;  Override`

Visibility: public

Description: `GetPreamble` overrides TEncoding.GetPreamble (1872) to return the 2 UTF-16 BOM Marker bytes ($FF,$FE).

See also: TEncoding.GetPreamble (1872)

76.111 TUNICODESTRINGBUILDER

76.111.1 Description

`TUnicodeStringBuilder` is a class to construct an ansistring out of other strings or characters. It's methods can be used instead of constructing the string manually using the normal + or concat operators. If used properly (e.g. by setting a sufficiently large Capacity (1955)), will result in faster operation. It offers various methods to append to the string.

To create a unicode string, use the TUnicodeStringBuilder (1949) class instead.

See also: TUnicodeStringBuilder (1949)

76.111.2 Method overview

Page	Method	Description
1950	Append	Append data in string form to the buffer.
1951	AppendFormat	Append the result of a `Format`.
1952	AppendLine	Append a string followed by a newline.
1952	Clear	Clear the string being built.
1952	CopyTo	Copy string bytes to an array.
1950	Create	Create a new `TUnicodeStringBuilder` instance.
1952	EnsureCapacity	Set the capacity if needed.
1953	Equals	Check if 2 stringbuilders have the same content string.
1953	Insert	Insert data in string form into the buffer at a given position.
1954	Remove	Remove data from the string.
1954	Replace	Replace a range of characters.
1955	ToString	Return the string buffer as an UnicodeString.

76.111.3 Property overview

Page	Properties	Access	Description
1955	Capacity	rw	Current capacity of the string buffer.
1955	Chars	rw	Indexed access to the characters in the string.
1955	Length	rw	Currentl length of the string buffer.
1956	MaxCapacity	r	Maximum capacity of the string buffer.

76.111.4 TUNICODESTRINGBUILDER.Create

Synopsis: Create a new `TUnicodeStringBuilder` instance.

Declaration:
```
constructor Create
constructor Create(aCapacity: Integer)
constructor Create(const AValue: UnicodeString)
constructor Create(aCapacity: Integer; aMaxCapacity: Integer)
constructor Create(const AValue: UnicodeString; aCapacity: Integer)
constructor Create(const AValue: UnicodeString; StartIndex: Integer;
                   aLength: Integer; aCapacity: Integer)
```

Visibility: public

Description: `Create` creates an empty `TUnicodeStringBuilder` instance. It can optionally set the initial capacity to `aCapacity` and the maximum capacity to `aMaxCapacity` and can also initialize the string buffer with characters from `aValue`: `aLength` characters starting at `StartIndex` (1-based) will be copied to the string buffer. If `aLength` and `StartIndex` are omitted, then all characters are copied.

See also: Capacity (1955), MaxCapacity (1956)

76.111.5 TUNICODESTRINGBUILDER.Append

Synopsis: Append data in string form to the buffer.

Declaration:
```
function Append(const AValue: Boolean) : TUNICODESTRINGBUILDER
function Append(const AValue: Byte) : TUNICODESTRINGBUILDER
function Append(const AValue: WideChar) : TUNICODESTRINGBUILDER
function Append(const AValue: Currency) : TUNICODESTRINGBUILDER
function Append(const AValue: Double) : TUNICODESTRINGBUILDER
```

```
function Append(const AValue: SmallInt) : TUNICODESTRINGBUILDER
function Append(const AValue: LongInt) : TUNICODESTRINGBUILDER
function Append(const AValue: Int64) : TUNICODESTRINGBUILDER
function Append(const AValue: TObject) : TUNICODESTRINGBUILDER
function Append(const AValue: ShortInt) : TUNICODESTRINGBUILDER
function Append(const AValue: Single) : TUNICODESTRINGBUILDER
function Append(const AValue: UInt64) : TUNICODESTRINGBUILDER
function Append(const AValue: Array of WideChar) : TUNICODESTRINGBUILDER
function Append(const AValue: Word) : TUNICODESTRINGBUILDER
function Append(const AValue: Cardinal) : TUNICODESTRINGBUILDER
function Append(const AValue: PWideChar) : TUNICODESTRINGBUILDER
function Append(const AValue: UnicodeString) : TUNICODESTRINGBUILDER
function Append(const AValue: RawByteString) : TUNICODESTRINGBUILDER
function Append(const AValue: WideChar; RepeatCount: Integer)
                 : TUNICODESTRINGBUILDER
function Append(const AValue: Array of WideChar; StartIndex: Integer;
               SBCharCount: Integer) : TUNICODESTRINGBUILDER
function Append(const AValue: UnicodeString; StartIndex: Integer;
               Count: Integer) : TUNICODESTRINGBUILDER
function Append(const Fmt: UnicodeString; const Args: Array of const)
                 : TUNICODESTRINGBUILDER
```

Visibility: public

Description: `Append` in its various overloaded forms will append the `aValue` argument to the buffer, after converting it to a string value if necessary:

- For integer values the various IntToSTr (1758) functions will be used.
- For TObject (1635) the TObject.ToString (1635) is used.
- The `repeatCount` argument will repeat the character as many times as instructed using StringOfChar (1635).
- If the `StartIndex` and `SBCharCount` are specified, then `SBCharCount` characters are copied starting from position `StartIndex`.

An overloaded version exists with the same arguments as AppendFormat (1951).

See also: StringOfChar (1635), AppendFormat (1951), AppendLine (1952), Insert (1953), Remove (1954)

76.111.6 TUNICODESTRINGBUILDER.AppendFormat

Synopsis: Append the result of a `Format`.

Declaration:
```
function AppendFormat(const Fmt: UnicodeString;
                      const Args: Array of const) : TUNICODESTRINGBUILDER
```

Visibility: public

Description: `AppendFormat` calls Format (1735) with `fmt` and `Args` and calls append (1950) with the resulting string.

Errors: the call to `Format` may result in an exception if the format parameters do not match the passed arguments.

See also: Append (1950), Insert (1953), Remove (1954), Format (1735)

76.111.7 TUNICODESTRINGBUILDER.AppendLine

Synopsis: Append a string followed by a newline.

Declaration:
```
function AppendLine : TUNICODESTRINGBUILDER
function AppendLine(const AValue: RawByteString) : TUNICODESTRINGBUILDER
```

Visibility: public

Description: `AppendLine` will append `aValue` (if it is specified) and appends the OS linebreak character (sLineBreak (1388)).

See also: TUnicodeStringBuilder.AppendFormat (1951), TUnicodeStringBuilder.Append (1950), #rtl.system.sLineBreak (1388), Remove (1954), Insert (1953)

76.111.8 TUNICODESTRINGBUILDER.Clear

Synopsis: Clear the string being built.

Declaration:
```
procedure Clear
```

Visibility: public

Description: `Clear` sets the length of the string to 0 and sets the Capacity (1955) to the default capacity (64 bytes).

See also: Capacity (1955), Remove (1954)

76.111.9 TUNICODESTRINGBUILDER.CopyTo

Synopsis: Copy string bytes to an array.

Declaration:
```
procedure CopyTo(SourceIndex: Integer;
                 var Destination: Array of WideChar;
                 DestinationIndex: Integer; Count: Integer)
```

Visibility: public

Description: `CopyTo` copies `aCount` characters from the string, starting at position `SourceIndex` (0-based) in the string to build, to the character array `Destination` at position `DestinationIndex`.

Errors: If the range of characters to copy falls outside the allowed range of characters in either source (available characters) or destination (available room for characters), then an ERangeError (1838) exception is raised.

See also: ERangeError (1838)

76.111.10 TUNICODESTRINGBUILDER.EnsureCapacity

Synopsis: Set the capacity if needed.

Declaration:
```
function EnsureCapacity(aCapacity: Integer) : Integer
```

Visibility: public

Description: `EnsureCapacity` checks that `aCapacity` is in the allowed range (less than MaxCapacity (1956)) and if the current capacity is less than `aCapacity`, increases the capacity to `aCapacity`.

Errors: If `aCapacity` is larger than MaxCapacity (1956) or negative, an ERangeError (1838) exception is raised.

See also: MaxCapacity (1956)

76.111.11 TUNICODESTRINGBUILDER.Equals

Synopsis: Check if 2 stringbuilders have the same content string.

Declaration:
```
function Equals(StringBuilder: TUNICODESTRINGBUILDER) : Boolean
                ;  Reintroduce
```

Visibility: public

Description: `Equals` checks that the self instance has the same content as the `StringBuilder` instance. It checks the length and content (as bytes) of the strings.

See also: TUnicodeStringBuilder.Clear (1952)

76.111.12 TUNICODESTRINGBUILDER.Insert

Synopsis: Insert data in string form into the buffer at a given position.

Declaration:
```
function Insert(Index: Integer; const AValue: Boolean)
                 : TUNICODESTRINGBUILDER
function Insert(Index: Integer; const AValue: Byte)
                 : TUNICODESTRINGBUILDER
function Insert(Index: Integer; const AValue: WideChar)
                 : TUNICODESTRINGBUILDER
function Insert(Index: Integer; const AValue: Currency)
                 : TUNICODESTRINGBUILDER
function Insert(Index: Integer; const AValue: Double)
                 : TUNICODESTRINGBUILDER
function Insert(Index: Integer; const AValue: SmallInt)
                 : TUNICODESTRINGBUILDER
function Insert(Index: Integer; const AValue: LongInt)
                 : TUNICODESTRINGBUILDER
function Insert(Index: Integer; const AValue: Array of WideChar)
                 : TUNICODESTRINGBUILDER
function Insert(Index: Integer; const AValue: Int64)
                 : TUNICODESTRINGBUILDER
function Insert(Index: Integer; const AValue: TObject)
                 : TUNICODESTRINGBUILDER
function Insert(Index: Integer; const AValue: ShortInt)
                 : TUNICODESTRINGBUILDER
function Insert(Index: Integer; const AValue: Single)
                 : TUNICODESTRINGBUILDER
function Insert(Index: Integer; const AValue: UnicodeString)
                 : TUNICODESTRINGBUILDER
function Insert(Index: Integer; const AValue: Word)
                 : TUNICODESTRINGBUILDER
function Insert(Index: Integer; const AValue: Cardinal)
                 : TUNICODESTRINGBUILDER
function Insert(Index: Integer; const AValue: UInt64)
                 : TUNICODESTRINGBUILDER
function Insert(Index: Integer; const AValue: UnicodeString;
                const aRepeatCount: Integer) : TUNICODESTRINGBUILDER
function Insert(Index: Integer; const AValue: Array of WideChar;
                startIndex: Integer; SBCharCount: Integer)
                 : TUNICODESTRINGBUILDER
```

Visibility: public

Description: `Insert` in its various overloaded forms will insert the `aValue` argument into the buffer at position `Index` (0-based), after converting it to a string value if necessary:

- For integer values the various IntToSTr (1758) functions will be used.
- For TObject (1635) the TObject.ToString (1635) is used.
- The `repeatCount` argument will insert the character as many times as instructed.
- If the `StartIndex` and `SBCharCount` are specified, then `SBCharCount` characters are copied starting from position `StartIndex`.

See also: AppendFormat (1951), AppendLine (1952), Append (1950), Remove (1954), Replace (1954)

76.111.13 TUNICODESTRINGBUILDER.Remove

Synopsis: Remove data from the string.

Declaration:
```
function Remove(StartIndex: Integer; RemLength: Integer)
                    : TUNICODESTRINGBUILDER
```

Visibility: public

Description: `Remove` will remove `RemLength` characters from the string buffer, starting at position `StartIndex` (0-based).

Errors: If the range of characters to remove falls outside the allowed range of characters, then an ERangeError (1838) exception is raised.

See also: AppendFormat (1951), AppendLine (1952), Append (1950), Insert (1953), Clear (1952)

76.111.14 TUNICODESTRINGBUILDER.Replace

Synopsis: Replace a range of characters.

Declaration:
```
function Replace(const OldChar: WideChar; const NewChar: WideChar)
                  : TUNICODESTRINGBUILDER
function Replace(const OldChar: WideChar; const NewChar: WideChar;
                 StartIndex: Integer; Count: Integer)
                  : TUNICODESTRINGBUILDER
function Replace(const OldValue: UnicodeString;
                 const NewValue: UnicodeString) : TUNICODESTRINGBUILDER
function Replace(const OldValue: UnicodeString;
                 const NewValue: UnicodeString; StartIndex: Integer;
                 Count: Integer) : TUNICODESTRINGBUILDER
```

Visibility: public

Description: `Replace` replaces occurrences of `OldValue` with the characters in `NewValue`. The search operation starts at position `StartIndex` (0-based, default 0) till position `StartIndex+aCount` is reached.

Errors: If `StartIndex` or `StartIndex+aCount` is out of range, an ERangeError (1838) exception is raised.

See also: AppendFormat (1951), AppendLine (1952), Append (1950), Insert (1953), Clear (1952), Remove (1954)

76.111.15 TUNICODESTRINGBUILDER.ToString

Synopsis: Return the string buffer as an UnicodeString.

Declaration:
```
function ToString : UnicodeString
function ToString(aStartIndex: Integer; aLength: Integer)
                  : UnicodeString;  Reintroduce
```

Visibility: public

Description: `ToString` returns the current content of the buffer as an UnicodeString.

See also: Clear (1952), CopyTo (1952)

76.111.16 TUNICODESTRINGBUILDER.Chars

Synopsis: Indexed access to the characters in the string.

Declaration:
```
Property Chars[index: Integer]: WideChar; default
```

Visibility: public

Access: Read,Write

Description: `Chars` allows indexed access to the characters in the string using a zero-based `Index`. The characters can be read or written. The allowed range for `Index` is 0 to Length (1955)-1.

See also: Length (1955)

76.111.17 TUNICODESTRINGBUILDER.Length

Synopsis: Currentl length of the string buffer.

Declaration:
```
Property &Length : Integer
```

Visibility: public

Access: Read,Write

Description: `Length` can be used to get or set the current length (in characters) of the string buffer. When setting the length, the new length is checked whether it lies within the maximum capacity (MaxCapacity (1956)) for the string buffer: if not, an ERangeError (1838) exception is raised. The capacity of the buffer will be adjusted so the buffer has anough capacity to accomodate the new length.

See also: MaxCapacity (1956), Capacity (1955), ERangeError (1838)

76.111.18 TUNICODESTRINGBUILDER.Capacity

Synopsis: Current capacity of the string buffer.

Declaration:
```
Property Capacity : Integer
```

Visibility: public

Access: Read,Write

Description: `Capacity` can be used to get or set the capacity of the string buffer (in characters). When setting the capacity, the new length is checked whether it lies within the maximum capacity (MaxCapacity (1956)) for the string buffer: if not, an ERangeError (1838) exception is raised.

See also: MaxCapacity (1956), Length (1955), ERangeError (1838)

76.111.19 TUNICODESTRINGBUILDER.MaxCapacity

Synopsis: Maximum capacity of the string buffer.

Declaration: `Property MaxCapacity : Integer`

Visibility: public

Access: Read

Description: `MaxCapacity` returns the maxmimum capacity of the string buffer (in characters). It is set when the stringbuilder is created, in the constructor of the builder. The Capacity (1955) (and consequently, the Length (1955)) will never be allowed to exceed `MaxCapacity`.

See also: Capacity (1955), Length (1955), ERangeError (1838)

76.112 TUTF7Encoding

76.112.1 Description

`TUTF7Encoding` is the encoding class used to represent the UTF7 encoding. This encoding is not so often used, and the class is provided mostly for completeness.

See also: TUnicodeEncoding (1948), TUTF8Encoding (1957), TMBCSEncoding (1894), TBigendianUnicodeEncoding (1851)

76.112.2 Method overview

Page	Method	Description
1956	Clone	Clone a `TUTF7Encoding` instance.
1956	Create	Create a new instance of the `TUTF7Encoding` class.
1957	GetMaxByteCount	Return max number of bytes needed to represent a string.
1957	GetMaxCharCount	Return max number of characters that can be represented by an array of bytes.

76.112.3 TUTF7Encoding.Create

Synopsis: Create a new instance of the `TUTF7Encoding` class.

Declaration: `constructor Create;  Override`

Visibility: public

Description: `Create` creates a new instance of the `TUTF7Encoding` class and sets the codepage to `CP_UTF7`.

See also: TEncoding.CodePage (1873), TEncoding (1869)

76.112.4 TUTF7Encoding.Clone

Synopsis: Clone a `TUTF7Encoding` instance.

Declaration: `function Clone : TEncoding;  Override`

Visibility: public

Description: `Clone` overrides TEncoding.Clone (1870) to provide a clone of the `TUTF7Encoding` instance.

See also: TEncoding.Clone (1870)

76.112.5 TUTF7Encoding.GetMaxByteCount

Synopsis: Return max number of bytes needed to represent a string.

Declaration: `function GetMaxByteCount(CharCount: Integer) : Integer; Override`

Visibility: public

Description: `GetMaxByteCount` overrides TEncoding.GetMaxByteCount (1872) to return the maximum number of bytes needed to represent a string.

See also: TEncoding.GetMaxByteCount (1872)

76.112.6 TUTF7Encoding.GetMaxCharCount

Synopsis: Return max number of characters that can be represented by an array of bytes.

Declaration: `function GetMaxCharCount(ByteCount: Integer) : Integer; Override`

Visibility: public

Description: `GetMaxCharCount` overrides TEncoding.GetMaxCharCount (1872) to return the maximum number of bytes needed to represent a string.

See also: TEncoding.GetMaxCharCount (1872)

76.113 TUTF8Encoding

76.113.1 Description

`TUTF8Encoding` is the encoding class used to represent the UTF-8 encoding.

See also: TUnicodeEncoding (1948), TUTF7Encoding (1956), TMBCSEncoding (1894), TBigendianUnicodeEncoding (1851)

76.113.2 Method overview

Page	Method	Description
1958	Clone	Clone a `TUTF8Encoding` instance.
1957	Create	Create a new instance of the `TUTF8Encoding` class.
1958	GetMaxByteCount	Return max number of bytes needed to represent a string.
1958	GetMaxCharCount	Return max number of characters that can be represented by an array of bytes.
1958	GetPreamble	Return BOM marker bytes.

76.113.3 TUTF8Encoding.Create

Synopsis: Create a new instance of the `TUTF8Encoding` class.

Declaration: `constructor Create; Override`

Visibility: public

Description: `Create` creates a new instance of the `TUTF8Encoding` class and sets the codepage to `CP_UTF8`.

See also: TEncoding.CodePage (1873), TEncoding (1869)

76.113.4 TUTF8Encoding.Clone

Synopsis: Clone a `TUTF8Encoding` instance.

Declaration: `function Clone : TEncoding;  Override`

Visibility: public

Description: `Clone` overrides TEncoding.Clone (1870) to provide a clone of the `TUTF8Encoding` instance.

See also: TEncoding.Clone (1870)

76.113.5 TUTF8Encoding.GetMaxByteCount

Synopsis: Return max number of bytes needed to represent a string.

Declaration: `function GetMaxByteCount(CharCount: Integer) : Integer;  Override`

Visibility: public

Description: `GetMaxByteCount` overrides TEncoding.GetMaxByteCount (1872) to return the maximum number of bytes needed to represent a string.

See also: TEncoding.GetMaxByteCount (1872)

76.113.6 TUTF8Encoding.GetMaxCharCount

Synopsis: Return max number of characters that can be represented by an array of bytes.

Declaration: `function GetMaxCharCount(ByteCount: Integer) : Integer;  Override`

Visibility: public

Description: `GetMaxCharCount` overrides TEncoding.GetMaxCharCount (1872) to return the maximum number of bytes needed to represent a string.

See also: TEncoding.GetMaxCharCount (1872)

76.113.7 TUTF8Encoding.GetPreamble

Synopsis: Return BOM marker bytes.

Declaration: `function GetPreamble : TBytes;  Override`

Visibility: public

Description: `GetPreamble` overrides TEncoding.GetPreamble (1872) to return the 3 UTF8 BOM Marker bytes ($EF,$BB,$BF).

See also: TEncoding.GetPreamble (1872)

76.114 TWordBoolHelper

76.114.1 Description

`TWordBoolHelper` is a helper type for the `WordBool` type. It contains mostly conversion routines to and from other types.

See also: TStringHelper (1926), TShortIntHelper (1910), TSmallIntHelper (1922), TWordHelper (1960), TCardinalHelper (1859), TIntegerHelper (1888), TInt64Helper (1884), TQWordHelper (1906), TNativeIntHelper (1898), TByteHelper (1855), TByteBoolHelper (1853), TWordBoolHelper (1959), TLongBoolHelper (1892)

76.114.2 Method overview

Page	Method	Description
1959	Parse	Convert string value to WordBool value.
1959	Size	Return the size (in bytes) of the.
1960	ToInteger	Convert to an integer value.
1959	ToString	Convert a WordBool value to string.
1960	TryToParse	Try to convert a string to a WordBool value.

76.114.3 TWordBoolHelper.Parse

Synopsis: Convert string value to WordBool value.

Declaration:
```
class function Parse(const S: string) : Boolean;  Static
```

Visibility: public

Description: `Parse` attempts to convert the string `S` to a WordBool value. It uses the StrToBool (1787) function to perform the conversion.

Errors: If `S` does not contain a valid string representation, then an EConvertError (1830) exception is raised.

See also: TWordBoolHelper.TryToParse (1960), TWordBoolHelper.ToString (1959), TWordBoolHelper.ToInteger (1960)

76.114.4 TWordBoolHelper.Size

Synopsis: Return the size (in bytes) of the.

Declaration:
```
class function Size : Integer;  Static
```

Visibility: public

Description: `Size` returns the size (in bytes) of the WordBool value. This is equivalent to `SizeOf(WordBool)`.

See also: SizeOf (1574)

76.114.5 TWordBoolHelper.ToString

Synopsis: Convert a WordBool value to string.

Declaration:
```
class function ToString(const AValue: Boolean;
                        UseBoolStrs: TUseBoolStrs) : string;  Overload
                        ;  Static
function ToString(UseBoolStrs: TUseBoolStrs) : string;  Overload
```

Visibility: public

Description: `ToString` will, in the class method version, convert the `AValue` WordBool to a string representation. In the function method version the WordBool value itself (`Self`) will be converted.

If the `UseBoolStrs` parameter equals `TUseBoolStrs.True`, then the string representation will use the WordBool strings BoolStrs (1635). The default value for `UseBoolStrs` is `TUseBoolStrs.False`.

The conversion is done using the BoolTostr (1685) function.

See also: BoolStrs (1635), BoolToStr (1685)

76.114.6 TWordBoolHelper.TryToParse

Synopsis: Try to convert a string to a WordBool value.

Declaration:
```
class function TryToParse(const S: string; out AValue: Boolean)
                          : Boolean;  Static
```

Visibility: public

Description: `TryToParse` will attempt to convert the string `S` to a WordBool value. If the attempt is successful, `True` is returned, and the actual value is returned in `AValue`. If the attempt failed, `False` is returned.

See also: TWordBoolHelper.Parse (1959), TWordBoolHelper.ToString (1959)

76.114.7 TWordBoolHelper.ToInteger

Synopsis: Convert to an integer value.

Declaration:
```
function ToInteger : Integer
```

Visibility: public

Description: `ToInteger` will return the WordBool value, typecasted to `Integer`.

See also: TWordBoolHelper.ToString (1959)

76.115 TWordHelper

76.115.1 Description

`TWordHelper` contains some auxiliary routines for a Word-typed ordinal value. It consists mainly of conversion routines to and from other types.

See also: TStringHelper (1926), TShortIntHelper (1910), TSmallIntHelper (1922), TByteHelper (1855), TCardinalHelper (1859), TIntegerHelper (1888), TInt64Helper (1884), TQWordHelper (1906), TNativeIntHelper (1898), TNativeUIntHelper (1902)

76.115.2 Method overview

Page	Method	Description
1964	ClearBit	Set bit to 0.
1961	Parse	Convert from a string.
1963	SetBit	Set bit to 1.
1961	Size	Size, in bytes, of the Word value.
1964	TestBit	Check bit.
1963	ToBinString	Convert to a binary string representation.
1962	ToBoolean	Convert to a boolean value.
1962	ToDouble	Convert to a double-sized floating point value.
1962	ToExtended	Convert to an extended-sized floating point value.
1964	ToggleBit	Invert bit.
1963	ToHexString	Convert to a hexadecimal string representation.
1963	ToSingle	Convert to an single-sized floating point value.
1961	ToString	Convert the value to string.
1962	TryParse	Try to convert a string to a Word, report success or failure.

76.115.3 TWordHelper.Parse

Synopsis: Convert from a string.

Declaration: `class function Parse(const AString: string) : Word;  Static`

Visibility: public

Description: `Parse` will attempt to convert the string `AString` to a Word value. It uses the StrToInt (1793) function to perform the conversion, so no localization is taken into account.

Errors: If the string does not contain a valid Word value, an EConvertError (1830) exception is raised.

See also: TWordHelper.ToString (1961), TWordHelper.TryParse (1962), StrToInt (1793)

76.115.4 TWordHelper.Size

Synopsis: Size, in bytes, of the Word value.

Declaration: `class function Size : Integer;  Static`

Visibility: public

Description: `Size` returns the size (in Words) of the Word value. This is equivalent to `SizeOf(Word)`.

Errors: None.

See also: SizeOf (1574)

76.115.5 TWordHelper.ToString

Synopsis: Convert the value to string.

Declaration: `class function ToString(const AValue: Word) : string;  Overload;  Static`
`function ToString : string;  Overload`

Visibility: public

Description: `ToString` will, in the class function variant of this method, convert `AValue` to a string representation. In the regular method overloaded version of `ToString`, the Word value itself is used. The IntToStr (1758) function is used to do the conversion.

See also: TWordHelper.Parse (1961), IntToStr (1758)

76.115.6 TWordHelper.TryParse

Synopsis: Try to convert a string to a Word, report success or failure.

Declaration:
```
class function TryParse(const AString: string; out AValue: Word)
                              : Boolean;  Static
```

Visibility: public

Description: `TryParse` attempts to convert the string `AString` to a Word, and reports the success of the attempt. If the attempt is successful, then `True` is returned, and the actual value of the Word is returned in `AValue`.

It uses the val (1635) function to perform the conversion, so no localization is taken into account.

See also: TWordHelper.Parse (1961), Val (1598)

76.115.7 TWordHelper.ToBoolean

Synopsis: Convert to a boolean value.

Declaration:
```
function ToBoolean : Boolean
```

Visibility: public

Description: `ToBoolean` converts the Word value to a boolean: it returns `True` if the value is nonzero, `False` if it is zero.

See also: TWordHelper.ToSingle (1963), TWordHelper.ToDouble (1962), TWordHelper.ToExtended (1962), TWordHelper.ToString (1961), TWordHelper.ToHexString (1963)

76.115.8 TWordHelper.ToDouble

Synopsis: Convert to a double-sized floating point value.

Declaration:
```
function ToDouble : Double
```

Visibility: public

Description: `ToDouble` converts the Word value to a double-sized floating point value.

See also: TWordHelper.ToBoolean (1962), TWordHelper.ToExtended (1962), TWordHelper.ToSingle (1963), TWordHelper.ToString (1961), TWordHelper.ToHexString (1963)

76.115.9 TWordHelper.ToExtended

Synopsis: Convert to an extended-sized floating point value.

Declaration:
```
function ToExtended : Extended
```

Visibility: public

Description: `ToDouble` converts the Word value to an extended-sized floating point value.

See also: TWordHelper.ToBoolean (1962), TWordHelper.ToSingle (1963), TWordHelper.ToDouble (1962), TWordHelper.ToString (1961), TWordHelper.ToHexString (1963)

76.115.10 TWordHelper.ToBinString

Synopsis: Convert to a binary string representation.

Declaration: `function ToBinString : string`

Visibility: public

Description: `ToBinString` converts the word value to a binary string representation, showing all 16 bits.

See also: TWordHelper.ToHexString (1963), TWordHelper.ToString (1961)

76.115.11 TWordHelper.ToHexString

Synopsis: Convert to a hexadecimal string representation.

Declaration: `function ToHexString(const AMinDigits: Integer) : string; Overload`
`function ToHexString : string; Overload`

Visibility: public

Description: `ToHexString` converts the Word value to a hexadecimal string representation. The `AMinDigits` argument specifies the minimal number of characters in the resulting string. The string will be left-padded with zeroes if the representation contains less than `AMinDigits` characters.

See also: TWordHelper.ToBoolean (1962), TWordHelper.ToSingle (1963), TWordHelper.ToDouble (1962), TWordHelper.ToString (1961), TWordHelper.ToExtended (1962)

76.115.12 TWordHelper.ToSingle

Synopsis: Convert to an single-sized floating point value.

Declaration: `function ToSingle : Single`

Visibility: public

Description: `ToSingle` converts the Word value to a single-sized floating point value.

See also: TWordHelper.ToBoolean (1962), TWordHelper.ToDouble (1962), TWordHelper.ToExtended (1962), TWordHelper.ToString (1961), TWordHelper.ToHexString (1963)

76.115.13 TWordHelper.SetBit

Synopsis: Set bit to 1.

Declaration: `function SetBit(const Index: TWordBitIndex) : Word`

Visibility: public

Description: `SetBit` puts value 1 to bit number determined by argument `Index`.

See also: TWordHelper.ClearBit (1964), TWordHelper.ToggleBit (1964), TWordHelper.TestBit (1964), TWordHelper.HighestSetBitPos (1960), TWordHelper.LowestSetBitPos (1960), TWordHelper.SetBitsCount (1960), TWordHelper.Bits (1960)

76.115.14 TWordHelper.ClearBit

Synopsis: Set bit to 0.

Declaration: `function ClearBit(const Index: TWordBitIndex) : Word`

Visibility: public

Description: `ClearBit` puts value 0 to bit number determined by argument `Index`.

See also: TWordHelper.SetBit (1963), TWordHelper.ToggleBit (1964), TWordHelper.TestBit (1964), TWordHelper.HighestSetBitPos (1960), TWordHelper.LowestSetBitPos (1960), TWordHelper.SetBitsCount (1960), TWordHelper.Bits (1960)

76.115.15 TWordHelper.ToggleBit

Synopsis: Invert bit.

Declaration: `function ToggleBit(const Index: TWordBitIndex) : Word`

Visibility: public

Description: `ToggleBit` reads bit value from bit number determined by argument `Index`, inverts it (if it was 0 it becomes 1, and vice versa), and stores it back.

See also: TWordHelper.SetBit (1963), TWordHelper.ToggleBit (1964), TWordHelper.TestBit (1964), TWordHelper.HighestSetBitPos (1960), TWordHelper.LowestSetBitPos (1960), TWordHelper.SetBitsCount (1960), TWordHelper.Bits (1960)

76.115.16 TWordHelper.TestBit

Synopsis: Check bit.

Declaration: `function TestBit(const Index: TWordBitIndex) : Boolean`

Visibility: public

Description: `TestBit` reads boolean value (true if bit is 1, and false if bit is 0) from bit number determined by argument `Index`.

See also: TWordHelper.SetBit (1963), TWordHelper.ToggleBit (1964), TWordHelper.TestBit (1964), TWordHelper.HighestSetBitPos (1960), TWordHelper.LowestSetBitPos (1960), TWordHelper.SetBitsCount (1960), TWordHelper.Bits (1960)

Chapter 77

Reference for unit 'Types'

77.1 Used units

Table 77.1: Used units by unit 'Types'

Name	Page
System	1362

77.2 Overview

Starting with D6, types from Windows specific units that were needed in Kylix were extracted to this unit. So it mostly contains type of Windows origin that are needed in the VCL framework.

77.3 Constants, types and variables

77.3.1 Constants

```
Epsilon : Single = 1E-40
```

```
Epsilon2 : Single = 1E-30
```

```
E_FAIL = HRESULT($80004005)
```

Defined for Delphi compatibility, this should not be used.

```
E_INVALIDARG = HRESULT($80070057)
```

Defined for Delphi compatibility, this should not be used.

```
GUID_NULL : TGuid = '{00000000-0000-0000-0000-000000000000}'
```

GUID_NULL is the definition of the NULL (empty) GUID.

```
LOCK_EXCLUSIVE = 2
```

Defined for Delphi compatibility, this should not be used.

```
LOCK_ONLYONCE = 4
```

Defined for Delphi compatibility, this should not be used.

```
LOCK_WRITE = 1
```

Defined for Delphi compatibility, this should not be used.

```
STATFLAG_DEFAULT = 0
```

Defined for Delphi compatibility, this should not be used.

```
STATFLAG_NONAME = 1
```

Defined for Delphi compatibility, this should not be used.

```
STATFLAG_NOOPEN = 2
```

Defined for Delphi compatibility, this should not be used.

```
STGTY_LOCKBYTES = 3
```

Defined for Delphi compatibility, this should not be used.

```
STGTY_PROPERTY = 4
```

Defined for Delphi compatibility, this should not be used.

```
STGTY_STORAGE = 1
```

Defined for Delphi compatibility, this should not be used.

```
STGTY_STREAM = 2
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_ABNORMALAPIEXIT = HRESULT($800300FA)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_ACCESSDENIED = HRESULT($80030005)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_BADBASEADDRESS = HRESULT($80030110)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_CANTSAVE = HRESULT($80030103)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_DISKISWRITEPROTECTED = HRESULT($80030013)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_DOCFILECORRUPT = HRESULT($80030109)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_EXTANTMARSHALLINGS = HRESULT($80030108)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_FILEALREADYEXISTS = HRESULT($80030050)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_FILENOTFOUND = HRESULT($80030002)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_INCOMPLETE = HRESULT($80030201)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_INSUFFICIENTMEMORY = HRESULT($80030008)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_INUSE = HRESULT($80030100)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_INVALIDFLAG = HRESULT($800300FF)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_INVALIDFUNCTION = HRESULT($80030001)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_INVALIDHANDLE = HRESULT($80030006)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_INVALIDHEADER = HRESULT($800300FB)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_INVALIDNAME = HRESULT($800300FC)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_INVALIDPARAMETER = HRESULT($80030057)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_INVALIDPOINTER = HRESULT($80030009)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_LOCKVIOLATION = HRESULT($80030021)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_MEDIUMFULL = HRESULT($80030070)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_NOMOREFILES = HRESULT($80030012)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_NOTCURRENT = HRESULT($80030101)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_OLDDLL = HRESULT($80030105)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_OLDFORMAT = HRESULT($80030104)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_PATHNOTFOUND = HRESULT($80030003)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_PROPSETMISMATCHED = HRESULT($800300F0)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_READFAULT = HRESULT($8003001E)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_REVERTED = HRESULT($80030102)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_SEEKERROR = HRESULT($80030019)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_SHAREREQUIRED = HRESULT($80030106)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_SHAREVIOLATION = HRESULT($80030020)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_TERMINATED = HRESULT($80030202)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_TOOMANYOPENFILES = HRESULT($80030004)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_UNIMPLEMENTEDFUNCTION = HRESULT($800300FE)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_UNKNOWN = HRESULT($800300FD)
```

Defined for Delphi compatibility, this should not be used.

```
STG_E_WRITEFAULT = HRESULT($8003001D)
```

Defined for Delphi compatibility, this should not be used.

```
STG_S_BLOCK = $00030201
```

Defined for Delphi compatibility, this should not be used.

```
STG_S_CONVERTED = $00030200
```

Defined for Delphi compatibility, this should not be used.

```
STG_S_MONITORING = $00030203
```

Defined for Delphi compatibility, this should not be used.

```
STG_S_RETRYNOW = $00030202
```

Defined for Delphi compatibility, this should not be used.

```
STREAM_SEEK_CUR = 1
```

Defined for Delphi compatibility, this should not be used.

```
STREAM_SEEK_END = 2
```

Defined for Delphi compatibility, this should not be used.

```
STREAM_SEEK_SET = 0
```

Defined for Delphi compatibility, this should not be used.

77.3.2 Types

`ArgList = Pointer`

`ArgList` is defined for Delphi/Kylix compatibility and should not be used.

`DWORD = LongWord`

Alias for cardinal type.

`FILETIME = _FILETIME`

Alias for the `_FILETIME` type.

`Largeint = Int64`

`Largeint` is an alias for the Int64 type defined in the system unit. This is an alias for Delphi/Kylix compatibility.

`LargeUint = QWord`

`LargeUInt` is an alias for the `QWord` type defined in the system unit. This is an alias for Delphi/Kylix compatibility.

`LARGE_INT = Largeint`

`LARGE_INT` is an alias for the Int64 type defined in the system unit. This is an alias for Delphi/Kylix compatibility.

`LARGE_UINT = LargeUint`

`LARGE_UINT` is an alias for the `QWord` type defined in the system unit. This is an alias for Delphi/Kylix compatibility.

`PByte = System.PByte`

`PByte` is defined in the system unit. This is an alias for Delphi/Kylix compatibility.

`PCLSID = PGuid`

`PCLSID` is a pointer to a `TCLSID` type.

`PDisplay = Pointer`

`PDisplay` is defined for Delphi/Kylix compatibility and should not be used.

`PDouble = System.PDouble`

`PDouble` is defined in the system unit. This is an alias for Delphi/Kylix compatibility.

`PDWord = ^DWORD`

PDWord is equivalent to the PCardinal type.

```
PEvent = Pointer
```

PEvent is defined for Delphi/Kylix compatibility and should not be used.

```
PFileTime = ^TFileTime
```

Pointer to TFileTime type.

```
PLargeInt = ^Largeint
```

PLargeInt is an alias for the PInt64 type defined in the system unit. This is an alias for Delphi/Kylix compatibility.

```
PLargeuInt = ^LargeUint
```

PLargeUInt is an alias for the PQWord type defined in the system unit. This is an alias for Delphi/Kylix compatibility.

```
PLongint = System.PLongint
```

PLongint is defined in the system unit. This is an alias for Delphi/Kylix compatibility.

```
POleStr = PWideChar
```

POleStr is a pointer to a (double) null-terminated array of TOleChar characters.

```
PPoint = ^TPoint
```

PPoint is a typed pointer to the TPoint (1983) type.

```
PPointF = ^TPointF
```

```
PPOleStr = ^POleStr
```

PPOleStr is a typed pointer to a POleStr variable.

```
PRect = ^TRect
```

PRect is a typed pointer to the TRect (1994) type.

```
PRectF = ^TRectF
```

```
PSize = ^TSize
```

PSize is a typed pointer to the TSize (2004) type.

```
PSizeF = ^TSizeF
```

```
PSmallInt = System.PSmallInt
```

PSmallInt is defined in the system unit. This is an alias for Delphi/Kylix compatibility.

```
PSmallPoint = ^TSmallPoint
```

PSmallPoint is a typed pointer to the TSmallPoint (1975) record.

```
PStatStg = ^TStatStg
```

Pointer to TStatStg record.

```
PXrmOptionDescRec = ^TXrmOptionDescRec
```

PXrmOptionDescRec is defined for Delphi/Kylix compatibility and should not be used.

```
Region = Pointer
```

Region is defined for Delphi/Kylix compatibility and should not be used.

```
STATSTG = TStatStg
```

Alias for the TStatStg type.

```
TagPoint = TPoint deprecated
```

tagPOINT is a simple alias for TPoint (1983)

```
TagSize = TSize deprecated
```

tagSize is an alias for the TSize (2004) type.

```
tagSTATSTG = record
public
  pwcsName : POleStr;
  dwType : DWORD
  ;
  cbSize : LARGE_UINT;
  mtime : TFileTime;
  ctime : TFileTime
  ;
  atime : TFileTime;
  grfMode : DWORD;
  grfLocksSupported : DWORD
  ;
  clsid : TCLSID;
  grfStateBits : DWORD;
  reserved : DWORD;
end
```

tagSTATSTG is used in the IStream.Stat (2014) call. It describes a storage medium (typically a file).

```
TAnsiStringDynArray = Array of AnsiString
```

```
TArray4IntegerType = Array[0..3] of LongInt
```

```
TBooleanDynArray = Array of Boolean
```

TBooleanDynArray is a standard definition of a dynamical array of booleans.

```
TByteDynArray = Array of Byte
```

TByteDynArray is a standard definition of a dynamical array of (8-bit, unsigned) bytes.

```
TCardinalDynArray = Array of Cardinal
```

TCardinalDynArray is a standard definition of a dynamical array of (32-bit, unsigned) cardinals.

```
TClassicByteDynArray = TByteDynArray
```

```
TClassicStringDynArray = TStringDynArray
```

```
TCLSID = TGuid
```

TCLSID is an alias for the #rtl.system.TGUID (1420) type.

```
TCompDynArray = Array of Comp
```

```
TCurrencyArray = Array of currency
```

```
TDirection = (FromBeginning,FromEnd)
```

Table 77.2: Enumeration values for type TDirection

Value	Explanation
FromBeginning	
FromEnd	

```
TDoubleDynArray = Array of Double
```

TSoubleDynArray is a standard definition of a dynamical array of doubles. (regular floating point type)

```
TDuplicates = (dupIgnore,dupAccept,dupError)
```

Table 77.3: Enumeration values for type TDuplicates

Value	Explanation
dupAccept	Accept duplicates, adding them to the list.
dupError	Raise an error when an attempt is made to add a duplicate.
dupIgnore	Ignore the new item, do not add it to the list.

`TDuplicates` can be used to indicate how a list structure acts on the addition of a duplicate item to the list.

dupIgnore Ignore the new item, do not add it to the list.

dupAccept Accept duplicates, adding them to the list.

dupError Raise an error when an attempt is made to add a duplicate.

```
TEndian = Objpas.TEndian
```

```
TExtendedDynArray = Array of Extended
```

```
TFileTime = _FILETIME
```

Alias for the `_FILETIME` type.

```
THorzRectAlign = (Center,Left,Right)
```

Table 77.4: Enumeration values for type THorzRectAlign

Value	Explanation
Center	
Left	
Right	

```
TInt64DynArray = Array of Int64
```

`TInt64DynArray` is a standard definition of a dynamical array of (64-bit, signed) int64s.

```
TIntegerDynArray = Array of Integer
```

`TIntegerDynArray` is a standard definition of a dynamical array of (32-bit, signed) integers.

```
TListCallback = procedure(data: pointer; arg: pointer) of object
```

`TListCallback` is the prototype for a `Foreach` operation on a list. It will be called with as `Data` the pointer in the list, and `Arg` will contain the extra user data added to the `Foreach` call. It can be used in methods of objects; for a version that can be used as a global procedure, see TListStaticCallback (1975)

```
TListStaticCallback = procedure(data: pointer; arg: pointer)
```

TListStaticCallback is the prototype for a Foreach operation on a list. It will be called with as Data the pointer in the list, and Arg will contain the extra user data added to the Foreach call. It can be used in plain procedures; for a version that can be used as a method, see TListCallback (1974)

```
TLongWordDynArray = Array of LongWord
```

TLongWordDynArray is a standard definition of a dynamical array of (32-bit, unsigned) LongWords.

```
TObjectDynArray = Array of TObject
```

```
TOleChar = WideChar
```

TOleChar is an alias for the WideChar type, defined in the system unit.

```
TPointerDynArray = Array of Pointer
```

Dynamic array of untyped pointers.

```
TQWordDynArray = Array of QWord
```

TQWordDynArray is a standard definition of a dynamical array of (64-bit, unsigned) QWords.

```
TRTLStringDynArray = Array of Ansistring
```

```
TShortIntDynArray = Array of ShortInt
```

TShortintDynArray is a standard definition of a dynamical array of (8-bit, signed) shortints.

```
TSingleDynArray = Array of Single
```

TSingleDynArray is a standard definition of a dynamical array of singles. (smallest floating point type)

```
TSmallIntDynArray = Array of SmallInt
```

TSmallintDynArray is a standard definition of a dynamical array of (16-bit, unsigned) integers.

```
TSmallPoint = packed record
public
  X : SmallInt;
  Y : SmallInt
  ;
end
```

TSmallPoint defines a point in a 2-dimensional plane, just like TPoint (1983), but the coordinates have a smaller range: The coordinates are smallints (16-bit, signed) and they run from -MaxSmallInt to maxSmallint.

```
TSplitRectType = (srLeft,srRight,srTop,srBottom)
```

Table 77.5: Enumeration values for type TSplitRectType

Value	Explanation
srBottom	
srLeft	
srRight	
srTop	

```
TStatStg = tagSTATSTG
```

`TStatStg` is a record type describing a storage medium. It is uses in the IStream.Stat (2014) function.

```
TStringDynArray = Array of AnsiString
```

`TStringDynArray` is a standard definition of a dynamical array of Ansistrings.

```
TUnicodeStringDynArray = Array of UnicodeString
```

```
TValueRelationship = - 1..1
```

```
TVertRectAlign = (Center,Top,Bottom)
```

Table 77.6: Enumeration values for type TVertRectAlign

Value	Explanation
Bottom	
Center	
Top	

```
TWideStringDynArray = Array of WideString
```

`TWideStringDynArray` is a standard definition of a dynamical array of WideStrings.

```
TWordDynArray = Array of Word
```

`TWordDynArray` is a standard definition of a dynamical array of (16-bit, unsigned) words.

```
TXrmOptionDescRec = record
end
```

`TXrmOptionDescRec` is defined for Delphi/Kylix compatibility and should not be used.

```
Widget = Pointer
```

`Widget` is defined for Delphi/Kylix compatibility and should not be used.

```
WidgetClass = Pointer
```

`WidgetClass` is defined for Delphi/Kylix compatibility and should not be used.

```
XrmOptionDescRec = TXrmOptionDescRec
```

`XrmOptionDescRec` is defined for Delphi/Kylix compatibility and should not be used.

```
_FILETIME = packed record
public
  dwLowDateTime : DWORD;
  dwHighDateTime
  : DWORD;
end
```

`_FILETIME` describes a file time stamp. It is defined or Delphi/Kylix compatibility and should not be used except when implementing or accessing the `IStream` interface. The `TDateTime` type should be used instead.

77.4 Procedures and functions

77.4.1 Bounds

Synopsis: Create a rectangle, given a position and size.

Declaration:
```
function Bounds(ALeft: Integer; ATop: Integer; AWidth: Integer;
                AHeight: Integer) : TRect
```

Visibility: default

Description: `Bounds` returns a `TRect` structure with the indicated position (`Left=ALeft` and `Top=ATop`) and size (`Right=ALeft+AWidth` and `Bottom=ATop+AHeight`)

See also: Rect (1980), PtInRect (1980), IntersectRect (1978), UnionRect (1981)

77.4.2 CenteredRect

Declaration:
```
function CenteredRect(const SourceRect: TRect;
                      const aCenteredRect: TRect) : TRect
```

Visibility: default

77.4.3 CenterPoint

Synopsis: Return the center point of a rectangle.

Declaration:
```
function CenterPoint(const Rect: TRect) : TPoint
```

Visibility: default

Description: `CenterPoint` returns the center point of the rectangle `Rect`.

See also: PtinRect (1980), IntersectRect (1978), IsRectEmpty (1979), OffsetRect (1979), InflateRect (1978), Size (1981), IsRectEmpty (1979)

77.4.4 EqualRect

Synopsis: Check if two rectangles are equal.

Declaration:
```
function EqualRect(const r1: TRect; const r2: TRect) : Boolean
function EqualRect(const r1: TRectF; const r2: TRectF) : Boolean
```

Visibility: default

Description: `EqualRect` returns `True` if the rectangles `R1` and `R2` are equal (i.e. have the position and size). If the rectangles differ, the function returns `False`

See also: Rect (1980), Bounds (1977), PtInRect (1980), IntersectRect (1978), UnionRect (1981), IsRectEmpty (1979), OffsetRect (1979), InflateRect (1978), Size (1981)

77.4.5 InflateRect

Synopsis: Increase the rectangle in size, keeping it centered.

Declaration:
```
function InflateRect(var Rect: TRect; dx: Integer; dy: Integer)
                          : Boolean
function InflateRect(var Rect: TRectF; dx: single; dy: Single) : Boolean
```

Visibility: default

Description: `InflateRect` inflates the rectangle horizontally with `dx` pixels on each side, and vertically with `dy` pixels, thus keeping its center point on the same location. It returns `true` if the operation was successfully, `False` if it was not (only possible if the address of `Rect` is `Nil`).

See also: PtinRect (1980), IntersectRect (1978), IsRectEmpty (1979), OffsetRect (1979), CenterPoint (1977), Size (1981), IsRectEmpty (1979)

77.4.6 IntersectRect

Synopsis: Return the intersection of 2 rectangles.

Declaration:
```
function IntersectRect(const Rect1: TRect; const Rect2: TRect) : Boolean
function IntersectRect(const Rect1: TRectF; const Rect2: TRectF)
                         : Boolean
function IntersectRect(var Rect: TRect; const R1: TRect;
                       const R2: TRect) : Boolean
function IntersectRect(var Rect: TRectF; const R1: TRectF;
                       const R2: TRectF) : Boolean
```

Visibility: default

Description: `IntersectRect` returns the intersection of the 2 rectangles `R1` and `R2` in `Rect`. It returns `True` if the 2 rectangles have an intersection, otherwise `False` is returned, and `Rect` is filled with zero.

See also: PtinRect (1980), UnionRect (1981), IsRectEmpty (1979), OffsetRect (1979), InflateRect (1978), Size (1981)

77.4.7 IntersectRectF

Declaration:
```
function IntersectRectF(out Rect: TRectF; const R1: TRectF;
                        const R2: TRectF) : Boolean
```

Visibility: default

77.4.8 IsRectEmpty

Synopsis: Check whether a rectangle is empty.

Declaration:
```
function IsRectEmpty(const Rect: TRectF) : Boolean
function IsRectEmpty(const Rect: TRect) : Boolean
```

Visibility: default

Description: `IsRectEmpty` returns true if the rectangle is empty, i.e. has a zero or negative width or height.

See also: PtinRect (1980), IntersectRect (1978), IsRectEmpty (1979), OffsetRect (1979), InflateRect (1978), Size (1981)

77.4.9 MinPoint

Declaration:
```
function MinPoint(const P1: TPointF; const P2: TPointF) : TPointF
                  ;  Overload
function MinPoint(const P1: TPoint; const P2: TPoint) : TPoint
                  ;  Overload
```

Visibility: default

77.4.10 MultiplyRect

Declaration:
```
procedure MultiplyRect(var R: TRectF; const DX: Single;
                       const DY: Single)
```

Visibility: default

77.4.11 NormalizeRect

Declaration:
```
function NormalizeRect(const ARect: TRectF) : TRectF;  Overload
```

Visibility: default

77.4.12 NormalizeRectF

Declaration:
```
function NormalizeRectF(const Pts: Array of TPointF) : TRectF;  Overload
```

Visibility: default

77.4.13 OffsetRect

Synopsis: Offset the rectangle.

Declaration:
```
function OffsetRect(var Rect: TRect; DX: Integer; DY: Integer) : Boolean
function OffsetRect(var Rect: TRectF; DX: Single; DY: Single) : Boolean
```

Visibility: default

Description: `OffsetRect` offsets the rectangle `Rect` by a horizontal distance `DX` and a vertical distance `DY`. The operation returns `True` if the operation was successful, `false` if it was not (only possible if the address of `Rect` is `Nil`).

See also: PtinRect (1980), IntersectRect (1978), IsRectEmpty (1979), OffsetRect (1979), InflateRect (1978), Size (1981), IsRectEmpty (1979)

77.4.14 Point

Synopsis: Create a point.

Declaration:
```
function Point(x: Integer; y: Integer) : TPoint
```

Visibility: default

Description: `Point` returns a `TPoint` structure with the given position (`X, Y`).

See also: Rect (1980), PtInRect (1980)

77.4.15 PointF

Declaration:
```
function PointF(x: Single; y: Single) : TPointF
```

Visibility: default

77.4.16 PtInRect

Synopsis: Check whether a point is inside a rectangle.

Declaration:
```
function PtInRect(const Rect: TRect; const p: TPoint) : Boolean
function PtInRect(const Rect: TRectF; const p: TPointF) : Boolean
```

Visibility: default

Description: `PtInRect` returns `True` if `p` is located inside `Rect`, and `False` if it is located outside the rectangle.

Remark Note that the bottom, right edges are not considered part of the rectangle, therefor a point located on one of these edges will not be considered part of the rectangle, meaning that for a record (10,10,100,100) the point (90,100) will not be considered part of the record, but `90,10` will be.

See also: IntersectRect (1978), UnionRect (1981), IsRectEmpty (1979), OffsetRect (1979), InflateRect (1978), Size (1981)

77.4.17 Rect

Synopsis: Create a rectangle record.

Declaration:
```
function Rect(Left: Integer; Top: Integer; Right: Integer;
              Bottom: Integer) : TRect
```

Visibility: default

Description: `Rect` returns a rectangle structure with the 4 members `Left`, `Top`, `Right` and `Bottom` as passed in the arguments.

See also: Bounds (1977), PtInRect (1980), IntersectRect (1978), UnionRect (1981), IsRectEmpty (1979), OffsetRect (1979), InflateRect (1978), Size (1981)

77.4.18 RectCenter

Declaration:
```
function RectCenter(var R: TRect; const Bounds: TRect) : TRect
function RectCenter(var R: TRectF; const Bounds: TRectF) : TRectF
```

Visibility: default

77.4.19 RectF

Declaration:
```
function RectF(Left: Single; Top: Single; Right: Single; Bottom: Single)
                    : TRectF
```

Visibility: default

77.4.20 RectHeight

Declaration:
```
function RectHeight(const Rect: TRect) : Integer
function RectHeight(const Rect: TRectF) : Single
```

Visibility: default

77.4.21 ScalePoint

Declaration:
```
function ScalePoint(const P: TPointF; dX: Single; dY: Single) : TPointF
                    ;  Overload
function ScalePoint(const P: TPoint; dX: Single; dY: Single) : TPoint
                    ;  Overload
```

Visibility: default

77.4.22 Size

Synopsis: Return the size of the rectangle.

Declaration:
```
function Size(AWidth: Integer; AHeight: Integer) : TSize
function Size(const ARect: TRect) : TSize
```

Visibility: default

Description: `Size` returns a `TSize` record with the indicated `AWidth, AHeight`. In the case `ARect` is passed, the width and height are calculated (taking into account that the right, bottom are not considered part of the rectangle).

See also: PtinRect (1980), IntersectRect (1978), IsRectEmpty (1979), OffsetRect (1979), InflateRect (1978), CenterPoint (1977), IsRectEmpty (1979)

77.4.23 SplitRect

Declaration:
```
function SplitRect(const Rect: TRect; SplitType: TSplitRectType;
                   Size: Integer) : TRect;  Overload
function SplitRect(const Rect: TRect; SplitType: TSplitRectType;
                   Percent: Double) : TRect;  Overload
```

Visibility: default

77.4.24 UnionRect

Synopsis: Return the union of 2 rectangles.

Declaration:
```
function UnionRect(var Rect: TRect; const R1: TRect; const R2: TRect)
                   : Boolean
function UnionRect(var Rect: TRectF; const R1: TRectF; const R2: TRectF)
                   : Boolean
function UnionRect(const R1: TRect; const R2: TRect) : TRect
function UnionRect(const R1: TRectF; const R2: TRectF) : TRectF
```

Visibility: default

Description: `UnionRect` retuns the rectangle that encompasses both `R1` and `R2` in `Rect`. It returns `True` if the resulting rectangle is not empty, `False` if the result is an empty rectangle (in which case the result is filled with zeroes)

See also: PtinRect (1980), IntersectRect (1978), IsRectEmpty (1979), OffsetRect (1979), InflateRect (1978), Size (1981)

77.4.25 UnionRectF

Declaration:
```
function UnionRectF(out Rect: TRectF; const R1: TRectF;
                    const R2: TRectF) : Boolean
```

Visibility: default

77.5 TPoint

```
TPoint = packed record
public
  X : LongInt;
  Y : LongInt;
  class
  function Zero : TPoint;  Static;
  function Add(const apt: TPoint
  ) : TPoint;
  function Distance(const apt: TPoint) : ValReal;
  function
  IsZero : Boolean;
  function Subtract(const apt: TPoint) : TPoint
  ;
  SetLocation;
  Offset;
  function Angle(const pt: TPoint) : Single
  ;
  class function PointInCircle(const apt: TPoint; const acenter
  : TPoint;
                               const aradius: Integer)
  : Boolean;  Static;
  TPoint.class operator =(const apt1: TPoint;
  const apt2: TPoint)
                          : Boolean;
  TPoint
  .class operator <>(const apt1: TPoint; const apt2: TPoint)
  : Boolean;
  TPoint.class operator +(const apt1
```

```
  : TPoint; const apt2: TPoint) : TPoint;
  TPoint.class operator -
  (const apt1: TPoint; const apt2: TPoint) : TPoint;
  TPoint.class
  operator :=(const aspt: TSmallPoint) : TPoint;
  TPoint.class operator
  explicit(const apt: TPoint) : TSmallPoint;
end
```

`TPoint` is a generic definition of a point in a 2-dimensional discrete plane, where `X` indicates the horizontal position, and `Y` the vertical position (positions usually measured in pixels), and `0,0` is the origin of the plane.

Usually, the origin is the upper-left corner of the screen, with Y increasing as one moves further down the screen - this is opposite to the mathematical view where Y increases as one moves upwards.

The coordinates are integers, (32-bit, signed) so the coordinate system runs from `-MaxInt` to `MaxInt`.

77.5.1 Method overview

Page	Method	Description
1983	Add	
1985	add(TPoint,TPoint):TPoint	
1984	Angle	
1985	assign(TSmallPoint):TPoint	
1983	Distance	
1984	equal(TPoint,TPoint):Boolean	
1985	explicit(TPoint):TSmallPoint	
1984	IsZero	
1984	notequal(TPoint,TPoint):Boolean	
1984	Offset	
1984	PointInCircle	
1984	SetLocation	
1984	Subtract	
1985	subtract(TPoint,TPoint):TPoint	
1983	Zero	

77.5.2 TPoint.Zero

Declaration: `class function Zero : TPoint;  Static`

Visibility: public

77.5.3 TPoint.Add

Declaration: `function Add(const apt: TPoint) : TPoint`

Visibility: public

77.5.4 TPoint.Distance

Declaration: `function Distance(const apt: TPoint) : ValReal`

Visibility: public

77.5.5 TPoint.IsZero

Declaration: `function IsZero : Boolean`

Visibility: public

77.5.6 TPoint.Subtract

Declaration: `function Subtract(const apt: TPoint) : TPoint`

Visibility: public

77.5.7 TPoint.SetLocation

Declaration:
```
procedure SetLocation(const apt: TPoint)
procedure SetLocation(ax: LongInt; ay: LongInt)
```

Visibility: public

77.5.8 TPoint.Offset

Declaration:
```
procedure Offset(const apt: TPoint)
procedure Offset(dx: LongInt; dy: LongInt)
```

Visibility: public

77.5.9 TPoint.Angle

Declaration: `function Angle(const pt: TPoint) : Single`

Visibility: public

77.5.10 TPoint.PointInCircle

Declaration:
```
class function PointInCircle(const apt: TPoint; const acenter: TPoint;
                             const aradius: Integer) : Boolean;  Static
```

Visibility: public

77.5.11 TPoint.equal(TPoint,TPoint):Boolean

Declaration:
```
TPoint.class operator =(const apt1: TPoint; const apt2: TPoint)
                       : Boolean
```

Visibility: public

77.5.12 TPoint.notequal(TPoint,TPoint):Boolean

Declaration:
```
TPoint.class operator <>(const apt1: TPoint; const apt2: TPoint)
                        : Boolean
```

Visibility: public

77.5.13 TPoint.add(TPoint,TPoint):TPoint

Declaration: `TPoint.class operator +(const apt1: TPoint; const apt2: TPoint) : TPoint`

Visibility: public

77.5.14 TPoint.subtract(TPoint,TPoint):TPoint

Declaration: `TPoint.class operator -(const apt1: TPoint; const apt2: TPoint) : TPoint`

Visibility: public

77.5.15 TPoint.assign(TSmallPoint):TPoint

Declaration: `TPoint.class operator :=(const aspt: TSmallPoint) : TPoint`

Visibility: public

77.5.16 TPoint.explicit(TPoint):TSmallPoint

Declaration: `TPoint.class operator explicit(const apt: TPoint) : TSmallPoint`

Visibility: public

77.6 TPoint3D

```
TPoint3D = packed record
public
  TSingle3Array = Array[0..2] of single
  ;
  constructor Create(const ax: single; const ay: single; const az
  : single);
  Offset;
case Integer of
0: (
public
  data : TSingle3Array
  ;
);
1: (
public
  x : single;
  y : single;
  z : single;
);
end
```

77.6.1 Method overview

Page	Method	Description
1986	Create	
1986	Offset	

77.6.2 TPoint3D.Create

Declaration: `constructor Create(const ax: single; const ay: single; const az: single)`

Visibility: public

77.6.3 TPoint3D.Offset

Declaration:
```
procedure Offset(const adeltax: single; const adeltay: single;
                 const adeltaz: single)
procedure Offset(const adelta: TPoint3D)
```

Visibility: public

77.7 TPointF

```
TPointF = packed record
public
  x : Single;
  y : Single;
  Add;
  function Distance(const apt: TPointF) : Single;
  function DotProduct
  (const apt: TPointF) : Single;
  function IsZero : Boolean;
  Subtract
  ;
  SetLocation;
  Offset;
  function EqualsTo(const apt: TPointF
  ; const aEpsilon: Single) : Boolean;
  function Scale(afactor: Single
  ) : TPointF;
  function Ceiling : TPoint;
  function Truncate : TPoint
  ;
  function Floor : TPoint;
  function Round : TPoint;
  function
  &Length : Single;
  function Rotate(angle: single) : TPointF;
  function
  Reflect(const normal: TPointF) : TPointF;
  function MidPoint(const
  b: TPointF) : TPointF;
  PointInCircle;
  class function Zero : TPointF
  ;  Static;
  function Angle(const b: TPointF) : Single;
  function
  AngleCosine(const b: TPointF) : single;
  function CrossProduct(const
  apt: TPointF) : Single;
```

```
  function Normalize : TPointF;
  Create
  ;
  TPointF.class operator =(const apt1: TPointF; const apt2: TPointF
  )
                            : Boolean;
  TPointF.class operator
  <>(const apt1: TPointF; const apt2: TPointF)
  : Boolean;
  TPointF.class operator +(const apt1: TPointF
  ; const apt2: TPointF)
                            : TPointF;
  TPointF
  .class operator -(const apt1: TPointF; const apt2: TPointF)
  : TPointF;
  TPointF.class operator -(const
  apt1: TPointF) : TPointF;
  TPointF.class operator *(const apt1:
  TPointF; const apt2: TPointF)
                            : TPointF
  ;
  TPointF.class operator *(const apt1: TPointF; afactor: single
  ) : TPointF;
  TPointF.class operator *(afactor: single; const apt1
  : TPointF) : TPointF;
  TPointF.class operator /(const apt1: TPointF
  ; afactor: single) : TPointF;
  TPointF.class operator :=(const apt
  : TPoint) : TPointF;
  TPointF.class operator **(const apt1: TPointF
  ; const apt2: TPointF)
                             : Single;
end
```

77.7.1 Method overview

Page	Method	Description
1988	Add	
1992	add(TPointF,TPointF):TPointF	
1991	Angle	
1991	AngleCosine	
1992	assign(TPoint):TPointF	
1989	Ceiling	
1991	Create	
1991	CrossProduct	
1988	Distance	
1992	divide(TPointF,single):TPointF	
1989	DotProduct	
1991	equal(TPointF,TPointF):Boolean	
1989	EqualsTo	
1990	Floor	
1989	IsZero	
1990	Length	
1990	MidPoint	
1992	multiply(single,TPointF):TPointF	
1992	multiply(TPointF,single):TPointF	
1992	multiply(TPointF,TPointF):TPointF	
1992	negative(TPointF):TPointF	
1991	Normalize	
1991	notequal(TPointF,TPointF):Boolean	
1989	Offset	
1990	PointInCircle	
1993	power(TPointF,TPointF):Single	
1990	Reflect	
1990	Rotate	
1990	Round	
1989	Scale	
1989	SetLocation	
1989	Subtract	
1992	subtract(TPointF,TPointF):TPointF	
1990	Truncate	
1991	Zero	

77.7.2 TPointF.Add

Declaration:
```
function Add(const apt: TPoint) : TPointF
function Add(const apt: TPointF) : TPointF
```

Visibility: public

77.7.3 TPointF.Distance

Declaration:
```
function Distance(const apt: TPointF) : Single
```

Visibility: public

77.7.4 TPointF.DotProduct

Declaration: `function DotProduct(const apt: TPointF) : Single`

Visibility: public

77.7.5 TPointF.IsZero

Declaration: `function IsZero : Boolean`

Visibility: public

77.7.6 TPointF.Subtract

Declaration:
```
function Subtract(const apt: TPointF) : TPointF
function Subtract(const apt: TPoint) : TPointF
```

Visibility: public

77.7.7 TPointF.SetLocation

Declaration:
```
procedure SetLocation(const apt: TPointF)
procedure SetLocation(const apt: TPoint)
procedure SetLocation(ax: Single; ay: Single)
```

Visibility: public

77.7.8 TPointF.Offset

Declaration:
```
procedure Offset(const apt: TPointF)
procedure Offset(const apt: TPoint)
procedure Offset(dx: Single; dy: Single)
```

Visibility: public

77.7.9 TPointF.EqualsTo

Declaration: `function EqualsTo(const apt: TPointF; const aEpsilon: Single) : Boolean`

Visibility: public

77.7.10 TPointF.Scale

Declaration: `function Scale(afactor: Single) : TPointF`

Visibility: public

77.7.11 TPointF.Ceiling

Declaration: `function Ceiling : TPoint`

Visibility: public

77.7.12 TPointF.Truncate

Declaration: `function Truncate : TPoint`

Visibility: public

77.7.13 TPointF.Floor

Declaration: `function Floor : TPoint`

Visibility: public

77.7.14 TPointF.Round

Declaration: `function Round : TPoint`

Visibility: public

77.7.15 TPointF.Length

Declaration: `function &Length : Single`

Visibility: public

77.7.16 TPointF.Rotate

Declaration: `function Rotate(angle: single) : TPointF`

Visibility: public

77.7.17 TPointF.Reflect

Declaration: `function Reflect(const normal: TPointF) : TPointF`

Visibility: public

77.7.18 TPointF.MidPoint

Declaration: `function MidPoint(const b: TPointF) : TPointF`

Visibility: public

77.7.19 TPointF.PointInCircle

Declaration:
```
class function PointInCircle(const pt: TPointF; const center: TPointF;
                             radius: single) : Boolean;  Static
class function PointInCircle(const pt: TPointF; const center: TPointF;
                             radius: Integer) : Boolean;  Static
```

Visibility: public

77.7.20 TPointF.Zero

Declaration: `class function Zero : TPointF;  Static`

Visibility: public

77.7.21 TPointF.Angle

Declaration: `function Angle(const b: TPointF) : Single`

Visibility: public

77.7.22 TPointF.AngleCosine

Declaration: `function AngleCosine(const b: TPointF) : single`

Visibility: public

77.7.23 TPointF.CrossProduct

Declaration: `function CrossProduct(const apt: TPointF) : Single`

Visibility: public

77.7.24 TPointF.Normalize

Declaration: `function Normalize : TPointF`

Visibility: public

77.7.25 TPointF.Create

Declaration:
```
class function Create(const ax: Single; const ay: Single) : TPointF
                        ;  Overload;  Static
class function Create(const apt: TPoint) : TPointF;  Overload;  Static
```

Visibility: public

77.7.26 TPointF.equal(TPointF,TPointF):Boolean

Declaration:
```
TPointF.class operator =(const apt1: TPointF; const apt2: TPointF)
                          : Boolean
```

Visibility: public

77.7.27 TPointF.notequal(TPointF,TPointF):Boolean

Declaration:
```
TPointF.class operator <>(const apt1: TPointF; const apt2: TPointF)
                          : Boolean
```

Visibility: public

77.7.28 TPointF.add(TPointF,TPointF):TPointF

Declaration: `TPointF.class operator +(const apt1: TPointF; const apt2: TPointF) : TPointF`

Visibility: public

77.7.29 TPointF.subtract(TPointF,TPointF):TPointF

Declaration: `TPointF.class operator -(const apt1: TPointF; const apt2: TPointF) : TPointF`

Visibility: public

77.7.30 TPointF.negative(TPointF):TPointF

Declaration: `TPointF.class operator -(const apt1: TPointF) : TPointF`

Visibility: public

77.7.31 TPointF.multiply(TPointF,TPointF):TPointF

Declaration: `TPointF.class operator *(const apt1: TPointF; const apt2: TPointF) : TPointF`

Visibility: public

77.7.32 TPointF.multiply(TPointF,single):TPointF

Declaration: `TPointF.class operator *(const apt1: TPointF; afactor: single) : TPointF`

Visibility: public

77.7.33 TPointF.multiply(single,TPointF):TPointF

Declaration: `TPointF.class operator *(afactor: single; const apt1: TPointF) : TPointF`

Visibility: public

77.7.34 TPointF.divide(TPointF,single):TPointF

Declaration: `TPointF.class operator /(const apt1: TPointF; afactor: single) : TPointF`

Visibility: public

77.7.35 TPointF.assign(TPoint):TPointF

Declaration: `TPointF.class operator :=(const apt: TPoint) : TPointF`

Visibility: public

77.7.36 TPointF.power(TPointF,TPointF):Single

Declaration: `TPointF.class operator **(const apt1: TPointF; const apt2: TPointF) : Single`

Visibility: public

77.8 TRect

```
TRect = packed record
private
  function getHeight : LongInt;
  function
  getLocation : TPoint;
  function getSize : TSize;
  function getWidth
  : LongInt;
  procedure setHeight(AValue: LongInt);
  procedure setSize
  (AValue: TSize);
  procedure setWidth(AValue: LongInt);
public
  Create
  ;
  TRect.class operator =(L: TRect; R: TRect) : Boolean;
  TRect
  .class operator <>(L: TRect; R: TRect) : Boolean;
  TRect.class operator
  +(L: TRect; R: TRect) : TRect;
  TRect.class operator *(L: TRect
  ; R: TRect) : TRect;
  class function Empty : TRect;  Static;
  procedure
  NormalizeRect;
  function IsEmpty : Boolean;
  Contains;
  function
  IntersectsWith(R: TRect) : Boolean;
  Intersect;
  Union;
  Offset
  ;
  SetLocation;
  Inflate;
  function CenterPoint : TPoint;
  SplitRect
  ;
  property Height : LongInt;
  property Width : LongInt;
  property
  Size : TSize;
  property Location : TPoint;
case LongInt of
```

```
0: (
  public
  Left : LongInt;
  Top : LongInt;
  Right : LongInt;
  Bottom
  : LongInt;
);
1: (
public
  TopLeft : TPoint;
  BottomRight : TPoint
  ;
);
2: (
public
  Vector : TArray4IntegerType;
);
end
```

`TRect` defines a rectangle in a discrete plane. It is described by the horizontal (`left`, `right`) or vertical (`top`, `Bottom`) positions (in pixels) of the edges, or, alternatively, by the coordinates of the top left (`TopLeft`) and bottom right (`BottomRight`) corners.

77.8.1 Method overview

Page	Method	Description
1995	add(TRect,TRect):TRect	
1996	CenterPoint	
1996	Contains	
1995	Create	
1995	Empty	
1995	equal(TRect,TRect):Boolean	
1996	Inflate	
1996	Intersect	
1996	IntersectsWith	
1995	IsEmpty	
1995	multiply(TRect,TRect):TRect	
1995	NormalizeRect	
1995	notequal(TRect,TRect):Boolean	
1996	Offset	
1996	SetLocation	
1997	SplitRect	
1996	Union	

77.8.2 Property overview

Page	Properties	Access	Description
1997	Height	rw	
1997	Location	rw	
1997	Size	rw	
1997	Width	rw	

77.8.3 TRect.Create

Declaration:
```
constructor Create(Origin: TPoint)
constructor Create(Origin: TPoint; AWidth: LongInt; AHeight: LongInt)
constructor Create(ALeft: LongInt; ATop: LongInt; ARight: LongInt;
                   ABottom: LongInt)
constructor Create(P1: TPoint; P2: TPoint; Normalize: Boolean)
constructor Create(R: TRect; Normalize: Boolean)
```

Visibility: public

77.8.4 TRect.equal(TRect,TRect):Boolean

Declaration:
```
TRect.class operator =(L: TRect; R: TRect) : Boolean
```

Visibility: public

77.8.5 TRect.notequal(TRect,TRect):Boolean

Declaration:
```
TRect.class operator <>(L: TRect; R: TRect) : Boolean
```

Visibility: public

77.8.6 TRect.add(TRect,TRect):TRect

Declaration:
```
TRect.class operator +(L: TRect; R: TRect) : TRect
```

Visibility: public

77.8.7 TRect.multiply(TRect,TRect):TRect

Declaration:
```
TRect.class operator *(L: TRect; R: TRect) : TRect
```

Visibility: public

77.8.8 TRect.Empty

Declaration:
```
class function Empty : TRect;  Static
```

Visibility: public

77.8.9 TRect.NormalizeRect

Declaration:
```
procedure NormalizeRect
```

Visibility: public

77.8.10 TRect.IsEmpty

Declaration:
```
function IsEmpty : Boolean
```

Visibility: public

77.8.11 TRect.Contains

Declaration:
```
function Contains(Pt: TPoint) : Boolean
function Contains(R: TRect) : Boolean
```

Visibility: public

77.8.12 TRect.IntersectsWith

Declaration:
```
function IntersectsWith(R: TRect) : Boolean
```

Visibility: public

77.8.13 TRect.Intersect

Declaration:
```
class function Intersect(R1: TRect; R2: TRect) : TRect;  Static
procedure Intersect(R: TRect)
```

Visibility: public

77.8.14 TRect.Union

Declaration:
```
class function Union(R1: TRect; R2: TRect) : TRect;  Static
procedure Union(R: TRect)
class function Union(const Points: Array of TPoint) : TRect;  Static
```

Visibility: public

77.8.15 TRect.Offset

Declaration:
```
procedure Offset(DX: LongInt; DY: LongInt)
procedure Offset(DP: TPoint)
```

Visibility: public

77.8.16 TRect.SetLocation

Declaration:
```
procedure SetLocation(X: LongInt; Y: LongInt)
procedure SetLocation(P: TPoint)
```

Visibility: public

77.8.17 TRect.Inflate

Declaration:
```
procedure Inflate(DX: LongInt; DY: LongInt)
procedure Inflate(DL: LongInt; DT: LongInt; DR: LongInt; DB: LongInt)
```

Visibility: public

77.8.18 TRect.CenterPoint

Declaration:
```
function CenterPoint : TPoint
```

Visibility: public

77.8.19 TRect.SplitRect

Declaration:
```
function SplitRect(SplitType: TSplitRectType; ASize: LongInt) : TRect
function SplitRect(SplitType: TSplitRectType; Percent: Double) : TRect
```

Visibility: public

77.8.20 TRect.Height

Declaration:
```
Property Height : LongInt
```

Visibility: public

Access: Read,Write

77.8.21 TRect.Width

Declaration:
```
Property Width : LongInt
```

Visibility: public

Access: Read,Write

77.8.22 TRect.Size

Declaration:
```
Property Size : TSize
```

Visibility: public

Access: Read,Write

77.8.23 TRect.Location

Declaration:
```
Property Location : TPoint
```

Visibility: public

Access: Read,Write

77.9 TRectF

```
TRectF = packed record
private
  function GetLocation : TPointF;
  function GetSize : TSizeF;
  procedure SetSize(AValue: TSizeF);
  function GetHeight : Single;
  function GetWidth : Single;
  procedure
  SetHeight(AValue: Single);
  procedure SetWidth(AValue: Single);
  public
  Create;
  TRectF.class operator =(L: TRectF; R: TRectF)
```

```
  : Boolean;
  TRectF.class operator <>(L: TRectF; R: TRectF) : Boolean
  ;
  TRectF.class operator +(L: TRectF; R: TRectF) : TRectF;
  TRectF
  .class operator *(L: TRectF; R: TRectF) : TRectF;
  TRectF.class operator
  :=(const arc: TRect) : TRectF;
  class function Empty : TRectF;
  Static;
  Intersect;
  Union;
  function Ceiling : TRectF;
  function
  CenterAt(const Dest: TRectF) : TRectF;
  function CenterPoint : TPointF
  ;
  Contains;
  function EqualsTo(const R: TRectF; const Epsilon:
  Single) : Boolean;
  function Fit(const Dest: TRectF) : Single;
  FitInto;
  function IntersectsWith(R: TRectF) : Boolean;
  function
  IsEmpty : Boolean;
  function PlaceInto(const Dest: TRectF;
  const AHorzAlign: THorzRectAlign;
  const AVertAlign: TVertRectAlign) : TRectF;
  function Round
  : TRect;
  function SnapToPixel(AScale: Single; APlaceBetweenPixels
  : Boolean)
                       : TRectF;
  function Truncate :
  TRect;
  Inflate;
  procedure NormalizeRect;
  Offset;
  SetLocation
  ;
  property Width : Single;
  property Height : Single;
  property
  Size : TSizeF;
  property Location : TPointF;
case Integer of
0:
  (
public
  Left : Single;
  Top : Single;
  Right : Single;
  Bottom
  : Single;
```

```
);
1: (
public
  TopLeft : TPointF;
  BottomRight : TPointF
  ;
);
end
```

77.9.1 Method overview

Page	Method	Description
2000	add(TRectF,TRectF):TRectF	
2000	assign(TRect):TRectF	
2001	Ceiling	
2001	CenterAt	
2001	CenterPoint	
2001	Contains	
1999	Create	
2000	Empty	
2000	equal(TRectF,TRectF):Boolean	
2001	EqualsTo	
2001	Fit	
2001	FitInto	
2002	Inflate	
2000	Intersect	
2002	IntersectsWith	
2002	IsEmpty	
2000	multiply(TRectF,TRectF):TRectF	
2002	NormalizeRect	
2000	notequal(TRectF,TRectF):Boolean	
2003	Offset	
2002	PlaceInto	
2002	Round	
2003	SetLocation	
2002	SnapToPixel	
2002	Truncate	
2001	Union	

77.9.2 Property overview

Page	Properties	Access	Description
2003	Height	rw	
2003	Location	rw	
2003	Size	rw	
2003	Width	rw	

77.9.3 TRectF.Create

Declaration:
```
constructor Create(Origin: TPointF)
constructor Create(Origin: TPointF; AWidth: Single; AHeight: Single)
constructor Create(ALeft: Single; ATop: Single; ARight: Single;
```

```
                    ABottom: Single)
constructor Create(P1: TPointF; P2: TPointF; Normalize: Boolean)
constructor Create(R: TRectF; Normalize: Boolean)
constructor Create(R: TRect; Normalize: Boolean)
```

Visibility: public

77.9.4 TRectF.equal(TRectF,TRectF):Boolean

Declaration: `TRectF.class operator =(L: TRectF; R: TRectF) : Boolean`

Visibility: public

77.9.5 TRectF.notequal(TRectF,TRectF):Boolean

Declaration: `TRectF.class operator <>(L: TRectF; R: TRectF) : Boolean`

Visibility: public

77.9.6 TRectF.add(TRectF,TRectF):TRectF

Declaration: `TRectF.class operator +(L: TRectF; R: TRectF) : TRectF`

Visibility: public

77.9.7 TRectF.multiply(TRectF,TRectF):TRectF

Declaration: `TRectF.class operator *(L: TRectF; R: TRectF) : TRectF`

Visibility: public

77.9.8 TRectF.assign(TRect):TRectF

Declaration: `TRectF.class operator :=(const arc: TRect) : TRectF`

Visibility: public

77.9.9 TRectF.Empty

Declaration: `class function Empty : TRectF;  Static`

Visibility: public

77.9.10 TRectF.Intersect

Declaration:
```
class function Intersect(R1: TRectF; R2: TRectF) : TRectF;  Static
procedure Intersect(R: TRectF)
```

Visibility: public

77.9.11 TRectF.Union

Declaration:
```
class function Union(const Points: Array of TPointF) : TRectF;  Static
class function Union(R1: TRectF; R2: TRectF) : TRectF;  Static
procedure Union(const r: TRectF)
```

Visibility: public

77.9.12 TRectF.Ceiling

Declaration:
```
function Ceiling : TRectF
```

Visibility: public

77.9.13 TRectF.CenterAt

Declaration:
```
function CenterAt(const Dest: TRectF) : TRectF
```

Visibility: public

77.9.14 TRectF.CenterPoint

Declaration:
```
function CenterPoint : TPointF
```

Visibility: public

77.9.15 TRectF.Contains

Declaration:
```
function Contains(Pt: TPointF) : Boolean
function Contains(R: TRectF) : Boolean
```

Visibility: public

77.9.16 TRectF.EqualsTo

Declaration:
```
function EqualsTo(const R: TRectF; const Epsilon: Single) : Boolean
```

Visibility: public

77.9.17 TRectF.Fit

Declaration:
```
function Fit(const Dest: TRectF) : Single
```

Visibility: public

77.9.18 TRectF.FitInto

Declaration:
```
function FitInto(const Dest: TRectF) : TRectF;  Overload
function FitInto(const Dest: TRectF; out Ratio: Single) : TRectF
                ;  Overload
```

Visibility: public

77.9.19 TRectF.IntersectsWith

Declaration:
```
function IntersectsWith(R: TRectF) : Boolean
```

Visibility: public

77.9.20 TRectF.IsEmpty

Declaration:
```
function IsEmpty : Boolean
```

Visibility: public

77.9.21 TRectF.PlaceInto

Declaration:
```
function PlaceInto(const Dest: TRectF;
                   const AHorzAlign: THorzRectAlign;
                   const AVertAlign: TVertRectAlign) : TRectF
```

Visibility: public

77.9.22 TRectF.Round

Declaration:
```
function Round : TRect
```

Visibility: public

77.9.23 TRectF.SnapToPixel

Declaration:
```
function SnapToPixel(AScale: Single; APlaceBetweenPixels: Boolean)
                     : TRectF
```

Visibility: public

77.9.24 TRectF.Truncate

Declaration:
```
function Truncate : TRect
```

Visibility: public

77.9.25 TRectF.Inflate

Declaration:
```
procedure Inflate(DL: Single; DT: Single; DR: Single; DB: Single)
procedure Inflate(DX: Single; DY: Single)
```

Visibility: public

77.9.26 TRectF.NormalizeRect

Declaration:
```
procedure NormalizeRect
```

Visibility: public

77.9.27 TRectF.Offset

Declaration:
```
procedure Offset(const dx: Single; const dy: Single)
procedure Offset(DP: TPointF)
```

Visibility: public

77.9.28 TRectF.SetLocation

Declaration:
```
procedure SetLocation(P: TPointF)
procedure SetLocation(X: Single; Y: Single)
```

Visibility: public

77.9.29 TRectF.Width

Declaration:
```
Property Width : Single
```

Visibility: public

Access: Read,Write

77.9.30 TRectF.Height

Declaration:
```
Property Height : Single
```

Visibility: public

Access: Read,Write

77.9.31 TRectF.Size

Declaration:
```
Property Size : TSizeF
```

Visibility: public

Access: Read,Write

77.9.32 TRectF.Location

Declaration:
```
Property Location : TPointF
```

Visibility: public

Access: Read,Write

77.10 TSize

```
TSize = packed record
public
  cx : LongInt;
  cy : LongInt;
  function
  Add(const asz: TSize) : TSize;
```

```
  function Distance(const asz: TSize
  ) : Double;
  function IsZero : Boolean;
  function Subtract(const
  asz: TSize) : TSize;
  TSize.class operator =(const asz1: TSize;
  const asz2: TSize) : Boolean;
  TSize.class operator <>(const asz1
  : TSize; const asz2: TSize) : Boolean;
  TSize.class operator +(const
  asz1: TSize; const asz2: TSize) : TSize;
  TSize.class operator
  -(const asz1: TSize; const asz2: TSize) : TSize;
  property Width
  : LongInt;
  property Height : LongInt;
end
```

`TSize` is a type to describe the size of a rectangular area, where `cx` is the width, `cy` is the height (in pixels) of the rectangle.

77.10.1 Method overview

Page	Method	Description
2004	Add	
2005	add(TSize,TSize):TSize	
2004	Distance	
2005	equal(TSize,TSize):Boolean	
2005	IsZero	
2005	notequal(TSize,TSize):Boolean	
2005	Subtract	
2005	subtract(TSize,TSize):TSize	

77.10.2 Property overview

Page	Properties	Access	Description
2005	Height	rw	
2005	Width	rw	

77.10.3 TSize.Add

Declaration: `function Add(const asz: TSize) : TSize`

Visibility: public

77.10.4 TSize.Distance

Declaration: `function Distance(const asz: TSize) : Double`

Visibility: public

77.10.5 TSize.IsZero

Declaration: `function IsZero : Boolean`

Visibility: public

77.10.6 TSize.Subtract

Declaration: `function Subtract(const asz: TSize) : TSize`

Visibility: public

77.10.7 TSize.equal(TSize,TSize):Boolean

Declaration: `TSize.class operator =(const asz1: TSize; const asz2: TSize) : Boolean`

Visibility: public

77.10.8 TSize.notequal(TSize,TSize):Boolean

Declaration: `TSize.class operator <>(const asz1: TSize; const asz2: TSize) : Boolean`

Visibility: public

77.10.9 TSize.add(TSize,TSize):TSize

Declaration: `TSize.class operator +(const asz1: TSize; const asz2: TSize) : TSize`

Visibility: public

77.10.10 TSize.subtract(TSize,TSize):TSize

Declaration: `TSize.class operator -(const asz1: TSize; const asz2: TSize) : TSize`

Visibility: public

77.10.11 TSize.Width

Declaration: `Property Width : LongInt`

Visibility: public

Access: Read,Write

77.10.12 TSize.Height

Declaration: `Property Height : LongInt`

Visibility: public

Access: Read,Write

77.11 TSizeF

```
TSizeF = packed record
public
  cx : Single;
  cy : Single;
  Add
  ;
  function Distance(const asz: TSizeF) : Single;
  function IsZero
  : Boolean;
  Subtract;
  function SwapDimensions : TSizeF;
  function
  Scale(afactor: Single) : TSizeF;
  function Ceiling : TSize;
  function
  Truncate : TSize;
  function Floor : TSize;
  function Round : TSize
  ;
  function &Length : Single;
  Create;
  TSizeF.class operator
  =(const asz1: TSizeF; const asz2: TSizeF)
  : Boolean;
  TSizeF.class operator <>(const asz1: TSizeF; const
  asz2: TSizeF)
                            : Boolean;
  TSizeF.class
  operator +(const asz1: TSizeF; const asz2: TSizeF) : TSizeF;
  TSizeF
  .class operator -(const asz1: TSizeF; const asz2: TSizeF) : TSizeF
  ;
  TSizeF.class operator -(const asz1: TSizeF) : TSizeF;
  TSizeF
  .class operator *(const asz1: TSizeF; afactor: single) : TSizeF;
  TSizeF.class operator *(afactor: single; const asz1: TSizeF) : TSizeF
  ;
  TSizeF.class operator :=(const apt: TPointF) : TSizeF;
  TSizeF
  .class operator :=(const asz: TSize) : TSizeF;
  TSizeF.class operator
  :=(const asz: TSizeF) : TPointF;
  property Width : Single;
  property
  Height : Single;
end
```

77.11.1 Method overview

Page	Method	Description
2007	Add	
2009	add(TSizeF,TSizeF):TSizeF	
2010	assign(TPointF):TSizeF	
2010	assign(TSize):TSizeF	
2010	assign(TSizeF):TPointF	
2008	Ceiling	
2009	Create	
2007	Distance	
2009	equal(TSizeF,TSizeF):Boolean	
2008	Floor	
2007	IsZero	
2008	Length	
2009	multiply(single,TSizeF):TSizeF	
2009	multiply(TSizeF,single):TSizeF	
2009	negative(TSizeF):TSizeF	
2009	notequal(TSizeF,TSizeF):Boolean	
2008	Round	
2008	Scale	
2008	Subtract	
2009	subtract(TSizeF,TSizeF):TSizeF	
2008	SwapDimensions	
2008	Truncate	

77.11.2 Property overview

Page	Properties	Access	Description
2010	Height	rw	
2010	Width	rw	

77.11.3 TSizeF.Add

Declaration:
```
function Add(const asz: TSize) : TSizeF
function Add(const asz: TSizeF) : TSizeF
```

Visibility: public

77.11.4 TSizeF.Distance

Declaration:
```
function Distance(const asz: TSizeF) : Single
```

Visibility: public

77.11.5 TSizeF.IsZero

Declaration:
```
function IsZero : Boolean
```

Visibility: public

77.11.6 TSizeF.Subtract

Declaration: `function Subtract(const asz: TSizeF) : TSizeF`
`function Subtract(const asz: TSize) : TSizeF`

Visibility: public

77.11.7 TSizeF.SwapDimensions

Declaration: `function SwapDimensions : TSizeF`

Visibility: public

77.11.8 TSizeF.Scale

Declaration: `function Scale(afactor: Single) : TSizeF`

Visibility: public

77.11.9 TSizeF.Ceiling

Declaration: `function Ceiling : TSize`

Visibility: public

77.11.10 TSizeF.Truncate

Declaration: `function Truncate : TSize`

Visibility: public

77.11.11 TSizeF.Floor

Declaration: `function Floor : TSize`

Visibility: public

77.11.12 TSizeF.Round

Declaration: `function Round : TSize`

Visibility: public

77.11.13 TSizeF.Length

Declaration: `function &Length : Single`

Visibility: public

77.11.14 TSizeF.Create

Declaration:
```
class function Create(const ax: Single; const ay: Single) : TSizeF
                      ;  Overload;  Static
class function Create(const asz: TSize) : TSizeF;  Overload;  Static
```

Visibility: public

77.11.15 TSizeF.equal(TSizeF,TSizeF):Boolean

Declaration:
```
TSizeF.class operator =(const asz1: TSizeF; const asz2: TSizeF)
                        : Boolean
```

Visibility: public

77.11.16 TSizeF.notequal(TSizeF,TSizeF):Boolean

Declaration:
```
TSizeF.class operator <>(const asz1: TSizeF; const asz2: TSizeF)
                         : Boolean
```

Visibility: public

77.11.17 TSizeF.add(TSizeF,TSizeF):TSizeF

Declaration:
```
TSizeF.class operator +(const asz1: TSizeF; const asz2: TSizeF) : TSizeF
```

Visibility: public

77.11.18 TSizeF.subtract(TSizeF,TSizeF):TSizeF

Declaration:
```
TSizeF.class operator -(const asz1: TSizeF; const asz2: TSizeF) : TSizeF
```

Visibility: public

77.11.19 TSizeF.negative(TSizeF):TSizeF

Declaration:
```
TSizeF.class operator -(const asz1: TSizeF) : TSizeF
```

Visibility: public

77.11.20 TSizeF.multiply(TSizeF,single):TSizeF

Declaration:
```
TSizeF.class operator *(const asz1: TSizeF; afactor: single) : TSizeF
```

Visibility: public

77.11.21 TSizeF.multiply(single,TSizeF):TSizeF

Declaration:
```
TSizeF.class operator *(afactor: single; const asz1: TSizeF) : TSizeF
```

Visibility: public

77.11.22 TSizeF.assign(TPointF):TSizeF

Declaration: `TSizeF.class operator :=(const apt: TPointF) : TSizeF`

Visibility: public

77.11.23 TSizeF.assign(TSize):TSizeF

Declaration: `TSizeF.class operator :=(const asz: TSize) : TSizeF`

Visibility: public

77.11.24 TSizeF.assign(TSizeF):TPointF

Declaration: `TSizeF.class operator :=(const asz: TSizeF) : TPointF`

Visibility: public

77.11.25 TSizeF.Width

Declaration: `Property Width : Single`

Visibility: public

Access: Read,Write

77.11.26 TSizeF.Height

Declaration: `Property Height : Single`

Visibility: public

Access: Read,Write

77.12 IClassFactory

77.12.1 Description

`IClassFactory` is defined for Delphi/Kylix compatibility and should not be used.

77.12.2 Method overview

Page	Method	Description
2010	CreateInstance	Create a new instance of an interface.
2011	LockServer	Lock ActiveX server object.

77.12.3 IClassFactory.CreateInstance

Synopsis: Create a new instance of an interface.

Declaration:
```
function CreateInstance(const unkOuter: IUnknown; const riid: TGuid;
                        out vObject) : HRESULT
```

Visibility: default

Description: `IClassFactory.CreateInstance` is defined for Delphi/Kylix compatibility and should not be used.

77.12.4 IClassFactory.LockServer

Synopsis: Lock ActiveX server object.

Declaration: `function LockServer(fLock: LongBool) : HRESULT`

Visibility: default

Description: `IClassFactory.LocksServer` is defined for Delphi/Kylix compatibility and should not be used.

77.13 ISequentialStream

77.13.1 Description

`ISequentialStream` is the interface for streams which only support sequential reading of chunks of data. It is defined for Delphi/Kylix compatibility and should not be used.

See also: IStream (2012)

77.13.2 Method overview

Page	Method	Description
2011	Read	Read data from the stream.
2011	Write	Write data to the stream.

77.13.3 ISequentialStream.Read

Synopsis: Read data from the stream.

Declaration: `function Read(pv: Pointer; cb: DWORD; pcbRead: PDWord) : HRESULT`

Visibility: default

Description: `Read` reads `cbCount` bytes from the stream into the memory pointed to by `pv` and returns the number of bytes read in `pcbread`. The result is zero for success or an error code.

See also: ISequentialStream.Write (2011)

77.13.4 ISequentialStream.Write

Synopsis: Write data to the stream.

Declaration: `function Write(pv: Pointer; cb: DWORD; pcbWritten: PDWord) : HRESULT`

Visibility: default

Description: `Write` writes `cbCount` bytes from the memory pointed to by `pv` to the stream and returns the number of bytes written in `pcbwritten`. The result is zero for success or an error code.

See also: ISequentialStream.Read (2011)

77.14 IStream

77.14.1 Description

An abstract interface for an external (non pascal) stream, as defined in Microsoft COM interfaces

77.14.2 Method overview

Page	Method	Description
2014	Clone	Clone the stream instance.
2013	Commit	Commit data to the stream.
2012	CopyTo	Copy data from one stream to another.
2013	LockRegion	Lock a region of bytes in the stream.
2013	Revert	Revert changes.
2012	Seek	Set the stream position.
2012	SetSize	Set the stream size.
2014	Stat	return information about the stream.
2013	UnlockRegion	Unlocks a previously locked region of bytes in the stream.

77.14.3 IStream.Seek

Synopsis: Set the stream position.

Declaration:
```
function Seek(dlibMove: Largeint; dwOrigin: DWORD;
                    out libNewPosition: LargeUint) : HRESULT
```

Visibility: default

Description: `Seek` sets the stream position at `dlibMove` bytes from `dwOrigin` (one of the `SEEK_*` constants) and returns the new absolute position in `libNewPosition`. The function returns zero on success, or an error code.

Errors: On error, a nonzero exit code is returned.

77.14.4 IStream.SetSize

Synopsis: Set the stream size.

Declaration:
```
function SetSize(libNewSize: LargeUint) : HRESULT
```

Visibility: default

Description: `SetSize` sets the size of the stream to `libNewSize` bytes, if the stream allows it. On success, zero is returned.

Errors: On error, a nonzero exit code is returned.

77.14.5 IStream.CopyTo

Synopsis: Copy data from one stream to another.

Declaration:
```
function CopyTo(stm: IStream; cb: LargeUint; out cbRead: LargeUint;
                    out cbWritten: LargeUint) : HRESULT
```

Visibility: default

Description: `CopyTo` copies `cb` bytes from the stream to target stream `stm`. `cbRead` returns how many bytes were read from the stream, `cbwrite` returns how many bytes were actually written to the destination stream. The function returns zero on success.

Errors: On error, a nonzero exit code is returned.

77.14.6 IStream.Commit

Synopsis: Commit data to the stream.

Declaration:
```
function Commit(grfCommitFlags: DWORD) : HRESULT
```

Visibility: default

Description: `Commit` commits the data in the stream to the underlying medium. Flags is a set of options to control the commit operation (see MSDN for the possible flags).

Errors: On error, a nonzero exit code is returned.

77.14.7 IStream.Revert

Synopsis: Revert changes.

Declaration:
```
function Revert : HRESULT
```

Visibility: default

Description: `Revert` reverts all changes that were done to a transacted stream, i.e. all changes since the last commit. The function returns zero on success.

Errors: On error, a nonzero exit code is returned.

77.14.8 IStream.LockRegion

Synopsis: Lock a region of bytes in the stream.

Declaration:
```
function LockRegion(libOffset: LargeUint; cb: LargeUint;
                    dwLockType: DWORD) : HRESULT
```

Visibility: default

Description: `LockRegion` locks a region of the storage, starting at `libOffset`, for `cbCount` bytes. The applied lock is of type `dwLockType`. The function returns zero if the lock was successfully applied.

Errors: On error, a nonzero exit code is returned.

77.14.9 IStream.UnlockRegion

Synopsis: Unlocks a previously locked region of bytes in the stream.

Declaration:
```
function UnlockRegion(libOffset: LargeUint; cb: LargeUint;
                      dwLockType: DWORD) : HRESULT
```

Visibility: default

Description: `UnlockRegion` removes the lock on a region of the storage, starting at `libOffset`, for `cbCount` bytes. The lock must be of type `dwLockType`. The function returns zero if the lock was successfully removed.

Errors: On error, a nonzero exit code is returned.

77.14.10 IStream.Stat

Synopsis: return information about the stream.

Declaration: `function Stat(out statstg: TStatStg; grfStatFlag: DWORD) : HRESULT`

Visibility: default

Description: `Stat` returns information about the stream in `statstg`, taking into account the flags in `grfStatFlag` (one of the `STATFLAG_` constants). The function returns zero if the call was successful.

Errors: On error, a nonzero exit code is returned.

77.14.11 IStream.Clone

Synopsis: Clone the stream instance.

Declaration: `function Clone(out stm: IStream) : HRESULT`

Visibility: default

Description: `Clone` returns an independent but initially equal copy of the stream in `stm`. The function returns zero if the call was successful.

Errors: On error, a nonzero exit code is returned.

77.15 TBitConverter

77.15.1 Method overview

Page	Method	Description
2014	From	
2015	InTo	
2014	UnsafeFrom	
2015	UnsafeInTo	

77.15.2 TBitConverter.UnsafeFrom

Declaration:
```
class procedure UnsafeFrom<T>(const ASrcValue: T;
                              var ADestination: Array of Byte;
                              AOffset: Integer);  Static
```

Visibility: default

77.15.3 TBitConverter.From

Declaration:
```
class procedure From<T>(const ASrcValue: T;
                        var ADestination: Array of Byte;
                        AOffset: Integer);  Static
```

Visibility: default

77.15.4 TBitConverter.UnsafeInTo

Declaration: `class function UnsafeInTo<T>(const ASource: Array of Byte; AOffset: Integer) : T;  Static`

Visibility: default

77.15.5 TBitConverter.InTo

Declaration: `class function InTo<T>(const ASource: Array of Byte; AOffset: Integer) : T;  Static`

Visibility: default

Chapter 78

Reference for unit 'TypInfo'

78.1 Used units

Table 78.1: Used units by unit 'TypInfo'

Name	Page
System	1362
sysutils	1635

78.2 Overview

The TypInfo unit contains many routines which can be used for the querying of the Run-Time Type Information (RTTI) which is generated by the compiler for classes that are compiled under the `{$M+}` switch. This information can be used to retrieve or set property values for published properties for totally unknown classes. In particular, it can be used to stream classes. The `TPersistent` class in the Classes unit is compiled in the `{$M+}` state and serves as the base class for all classes that need to be streamed.

The unit should be compatible to the Delphi unit with the same name.

The examples in this chapter use a rttiobj auxiliary unit, which contains an object that has a published property for all supported types. It also contains some auxiliary routines and definitions. This unit is included in the documentation sources, in the directory typinfex.

78.3 Auxiliary functions.

Other typinfo related functions.

Table 78.2:

Name	Description
GetEnumName (2032)	Get an enumerated type element name
GetEnumValue (2034)	Get ordinal number of an enumerated type, based on the name.
GetEnumNameCount (2033)	Get number of elements in an enumerated type.
GetTypeData (2047)	Skip type name and return a pointer to the type data
SetToString (2057)	Convert a set to its string representation
StringToSet (2059)	Convert a string representation of a set to a set

78.4 Getting or setting property values.

Functions to set or set a property's value.

Table 78.3:

Name	Description
GetEnumProp (2033)	Return the value of an enumerated type property
GetFloatProp (2034)	Return the value of a float property
GetInt64Prop (2035)	Return the value of an Int64 property
GetMethodProp (2036)	Return the value of a procedural type property
GetObjectProp (2038)	Return the value of an object property
GetOrdProp (2040)	Return the value of an ordinal type property
GetPropValue (2043)	Return the value of a property as a variant
GetSetProp (2044)	Return the value of a set property
GetStrProp (2046)	Return the value of a string property
GetWideStrProp (2048)	Return the value of a widestring property
GetVariantProp (2047)	Return the value of a variant property
SetEnumProp (2053)	Set the value of an enumerated type property
SetFloatProp (2053)	Set the value of a float property
SetInt64Prop (2053)	Set the value of an Int64 property
SetMethodProp (2054)	Set the value of a procedural type property
SetObjectProp (2055)	Set the value of an object property
SetOrdProp (2055)	Set the value of an ordinal type property
SetPropValue (2056)	Set the value of a property through a variant
SetSetProp (2057)	Set the value of a set property
SetStrProp (2057)	Set the value of a string property
SetWideStrProp (2059)	Set the value of a widestring property
SetVariantProp (2059)	Set the value of a variant property

78.5 Examining published property information.

Functions for retrieving or examining property information

Table 78.4:

Name	Description
FindPropInfo (2030)	Getting property type information, With error checking.
GetPropInfo (2041)	Getting property type information, No error checking.
GetPropInfos (2041)	Find property information of a certain kind
GetObjectPropClass (2039)	Return the declared class of an object property
GetPropList (2042)	Get a list of all published properties
IsPublishedProp (2048)	Is a property published
IsStoredProp (2049)	Is a property stored
PropIsType (2050)	Is a property of a certain kind
PropType (2051)	Return the type of a property

78.6 Constants, types and variables

78.6.1 Constants

```
BooleanIdents : Array[Boolean] of string = ('False', 'True')
```

Names for boolean values.

```
DotSep : string = '.'
```

Name separator character.

```
OnGetPropValue : TGetPropValue = Nil
```

This callback is set by the variants unit to enable reading of properties as a variant. If set, it is called by the GetPropValue (2043) function.

```
OnGetVariantprop : TGetVariantProp = Nil
```

This callback is set by the variants unit to enable reading of variant properties If set, it is called by the GetVariantProp (2047) function.

```
OnSetPropValue : TSetPropValue = Nil
```

This callback is set by the variants unit to enable writing of properties as a variant. If set, it is called by the SetPropValue (2056) function.

```
OnSetVariantprop : TSetVariantProp = Nil
```

This callback is set by the variants unit to enable writing of variant properties. If set, it is called by the GetVariantProp (2047) function.

```
ptConst = 3
```

Constant used in access method.

```
ptField = 0
```

Property access directly from field.

```
ptStatic = 1
```

Property access via static method.

```
ptVirtual = 2
```

Property access via virtual method.

```
tkAny = [Low(TTypeKind)..High(TTypeKind)]
```

Any property type.

```
tkArray = System.tkArray
```

Array property.

```
tkAString = System.tkAString
```

Ansistring property.

```
tkBool = System.tkBool
```

Boolean property.

```
tkChar = System.tkChar
```

Char property.

```
tkClass = System.tkClass
```

Class property.

```
tkClassRef = System.tkClassRef
```

```
tkDynArray = System.tkDynArray
```

Dynamical array property.

```
tkEnumeration = System.tkEnumeration
```

Enumeration type property.

```
tkFile = System.tkFile
```

```
tkFloat = System.tkFloat
```

Float property.

```
tkHelper = System.tkHelper
```

```
tkInt64 = System.tkInt64
```

Int64 property.

```
tkInteger = System.tkInteger
```

Integer property.

```
tkInterface = System.tkInterface
```

Interface property.

```
tkInterfaceRaw = System.tkInterfaceRaw
```

Raw interface property.

```
tkLString = System.tkLString
```

Longstring property.

```
tkMethod = System.tkMethod
```

Method property.

```
tkMethods = [tkMethod]
```

Only method properties. (event handlers).

```
tkObject = System.tkObject
```

Object property.

```
tkPointer = System.tkPointer
```

```
tkProcedure = tkProcVar
```

Procedure kind.

```
tkProcVar = System.tkProcVar
```

```
tkProperties = tkAny - tkMethods - [tkUnknown]
```

Real properties. (not methods).

```
tkQWord = System.tkQWord
```

QWord property.

```
tkRecord = System.tkRecord
```

Record property.

```
tkSet = System.tkSet
```

Set property.

```
tkSString = System.tkSString
```

Shortstring property.

```
tkString = tkSString
```

Alias for the `tsSString` enumeration value.

```
tkUChar = System.tkUChar
```

```
tkUnknown = System.tkUnknown
```

Unknown property type.

```
tkUString = System.tkUString
```

```
tkVariant = System.tkVariant
```

Variant property.

```
tkWChar = System.tkWChar
```

Widechar property.

```
tkWString = System.tkWString
```

Widestring property.

78.6.2 Types

```
PClassData = ^TClassData
```

`PClassData` is a pointer to a TClassData (2061) record.

```
PInitManagedField = ^TInitManagedField
```

`PInitManagedField` is a pointer to a TInitManagedField (2024) record.

```
PInterfaceData = ^TInterfaceData
```

`PInterfaceData` is a pointer to a TInterfaceData (2062) record.

```
PInterfaceRawData = ^TInterfaceRawData
```

Pointer to a `TInterfaceRawData` record.

```
PIntfMethodEntry = ^TIntfMethodEntry
```

`PIntfMethodEntry` is a pointer to a TIntfMethodEntry (2065) record.

```
PIntfMethodTable = ^TIntfMethodTable
```

`PIntfMethodTable` is a pointer to a TIntfMethodTable (2067) record.

```
PManagedField = ^TManagedField
```

`PManagedField` is a pointer to TManagedField (2068). It is used to describe automatically managed fields in records when the type kind is `tkRecord`.

```
PParameterLocation = ^TParameterLocation
```

`PParameterLocation` defines a pointer to a TParameterLocation (2069) record.

```
PParameterLocations = ^TParameterLocations
```

`PParameterLocations` points to a TParameterLocations (2070) record.

```
PProcedureParam = ^TProcedureParam
```

`PProcedureParam` is a pointer to `TProcedureParam`. It is used in TProcedureSignature (2072).

```
PPropData = ^TPropData
```

`PPropData` is a pointer to a TPropData (2073) record.

```
PPropInfo = ^TPropInfo
```

Pointer to TPropInfo (2075) record.

```
PPropList = ^TPropList
```

Pointer to TPropList (2026).

```
PPTypeInfo = ^PTypeInfo
```

Pointer to PTypeInfo (2023) pointer.

```
PRecInitData = ^TRecInitData
```

`PRecInitData` is a pointer to a TRecInitData (2027) record.

```
PRecOpOffsetTable = ^TRecOpOffsetTable
```

PRecOpOffsetTable is a Pointer to a TRecOpOffsetTable (2027) record.

```
PTypeData = ^TTypeData
```

Pointer to TTypeData (2091) record.

```
PTypeInfo = ^TTypeInfo
```

Pointer to TTypeInfo (2028) record.

```
PVmtFieldClassTab = ^TVmtFieldClassTab
```

PVmtFieldClassTab points to a TVmtFieldClassTab (2029) record.

```
PVmtFieldEntry = ^TVmtFieldEntry
```

Pointer to #rtl.typinfo.TVmtFieldEntry (2094) type.

```
PVmtFieldTable = ^TVmtFieldTable
```

Pointer to #rtl.typinfo.TVmtFieldTable (2095) type.

```
PVmtMethodEntry = ^TVmtMethodEntry
```

PVmtMethodEntry is a Pointer to TVmtMethodEntry (2029) record.

```
PVmtMethodParam = ^TVmtMethodParam
```

PVmtMethodParam is a pointer to a TVmtMethodParam (2096) record.

```
PVmtMethodTable = ^TVmtMethodTable
```

PVmtMethodTable is a pointer to a TVmtMethodTable (2097) record.

```
ShortStringBase = string
```

ShortStringBase is the base definition of a short string.

```
TCallConv = (ccReg,ccCdecl,ccPascal,ccStdCall,ccSafeCall,ccCppdecl
  ,
             ccFar16,ccOldFPCCall,ccInternProc,ccSysCall,ccSoftFloat
  ,
             ccMWPascal)
```

Table 78.5: Enumeration values for type TCallConv

Value	Explanation
ccCdecl	Cdecl calling convention.
ccCppdecl	Cppdecl calling convention.
ccFar16	Far16 calling convention (Delphi compatibility).
ccInternProc	InternProc calling convention (compiler internal).
ccMWPascal	MWPascal (MetroWerks Pascal) calling convention.
ccOldFPCCall	OldFPCCall calling convention (deprecated).
ccPascal	Pascal calling convention.
ccReg	Register calling convention.
ccSafeCall	SafeCall calling convention.
ccSoftFloat	Softfloat calling convention.
ccStdCall	stdcall calling convention.
ccSysCall	SysCall calling convention.

`TCallConv` is a type describing the calling convention used by a method. It contains an element for all supported calling conventions.

```
TFloatType = (ftSingle,ftDouble,ftExtended,ftComp,ftCurr)
```

Table 78.6: Enumeration values for type TFloatType

Value	Explanation
ftComp	Comp-type float.
ftCurr	Currency-type float.
ftDouble	Double-sized float.
ftExtended	Extended-size float.
ftSingle	Single-sized float.

The size of a float type.

```
TGetPropValue = function(Instance: TObject; PropInfo: PPropInfo;
  PreferStrings: Boolean) : Variant
```

The callback function must return the property with name `PropName` of instance `Instance`. If `PreferStrings` is true, it should favour converting the property to a string value. The function needs to return the variant with the property value.

```
TGetVariantProp = function(Instance: TObject; PropInfo: PPropInfo
  )
                               : Variant
```

The callback function must return the variant property with name `PropName` of instance `Instance`.

```
TInitManagedField = TManagedField
```

`TInitManagedField` is an alias for TManagedField (2068).

```
TIntfFlag = (ifHasGuid,ifDispInterface,ifDispatch,ifHasStrGUID)
```

Table 78.7: Enumeration values for type TIntfFlag

Value	Explanation
ifDispatch	Interface is a dispatch interface.
ifDispInterface	Interface is a dual dispatch interface.
ifHasGuid	Interface has GUID identifier.
ifHasStrGUID	Interface has a string GUID identifier.

Type of interface.

```
TIntfFlags = Set of TIntfFlag
```

Set of TIntfFlag (2025).

```
TIntfFlagsBase = Set of TIntfFlag
```

Set of TIntfFlag (2025).

```
TMethodKind = (mkProcedure,mkFunction,mkConstructor,mkDestructor,
  mkClassProcedure,mkClassFunction,mkClassConstructor
  ,
               mkClassDestructor,mkOperatorOverload)
```

Table 78.8: Enumeration values for type TMethodKind

Value	Explanation
mkClassConstructor	Class constructor method.
mkClassDestructor	Class destructor method.
mkClassFunction	Class function.
mkClassProcedure	Class procedure.
mkConstructor	Class constructor.
mkDestructor	Class Destructor.
mkFunction	Function method.
mkOperatorOverload	Operator overloader.
mkProcedure	Procedure method.

Method type description.

```
TOrdType = (otSByte,otUByte,otSWord,otUWord,otSLong,otULong,otSQWord
  ,
            otUQWord)
```

Table 78.9: Enumeration values for type TOrdType

Value	Explanation
otSByte	Signed byte.
otSLong	Signed longint.
otSQWord	Signed QWord.
otSWord	Signed word.
otUByte	Unsigned byte.
otULong	Unsigned longing (Cardinal).
otUQWord	Unsigned QWord.
otUWord	Unsigned word.

If the property is and ordinal type, then `TOrdType` determines the size and sign of the ordinal type:

```
TParamFlag = (pfVar,pfConst,pfArray,pfAddress,pfReference,pfOut,
  pfConstRef,pfHidden,pfHigh,pfSelf,pfVmt,pfResult)
```

Table 78.10: Enumeration values for type TParamFlag

Value	Explanation
pfAddress	Parameter is passed by address.
pfArray	Parameter is an array parameter.
pfConst	Parameter is a const parameter (i.e. cannot be modified).
pfConstRef	Constref parameter type.
pfHidden	hidden parameter type.
pfHigh	High bound parameter type.
pfOut	Parameter is a string parameter.
pfReference	Parameter is passed by reference.
pfResult	Result location parameter.
pfSelf	Self parameter type.
pfVar	Parameter is a var parameter (passed by reference).
pfVmt	VMT parameter type.

`TParamFlag` describes a parameter.

```
TParamFlags = Set of TParamFlag
```

The kind of parameter for a method.

```
TProcInfoProc = procedure(PropInfo: PPropInfo) of object
```

Property info callback method.

```
TPropList = Array[0..65535] of PPropInfo
```

Array of property information pointers.

```
TRecInitData = packed record
```

```
public
  Terminator : Pointer;
  Size
  : Integer;
  InitOffsetOp : PRecOpOffsetTable;
  ManagementOp : Pointer
  ;
  ManagedFieldCount : Integer;
end
```

`TRecInitData` describes the generated RTTI info with which you can initialize a record that contains managed fields.

```
TRecOpOffsetEntry = packed record
public
  ManagementOperator : CodePointer
  ;
  FieldOffset : SizeUInt;
end
```

`TRecOpOffsetEntry` describes management operator info for a record.

```
TRecOpOffsetTable = packed record
public
  Count : LongWord;
  Entries
  : Array[0..0] of TRecOpOffsetEntry;
end
```

`TRecOpOffsetTable` is a table of TRecOpOffsetEntry (2027) records with management operator info for the managed fields of a record.

```
TRegisterType = (Invalid,Int,FP,MMX,MultiMedia,Special,Address)
```

Table 78.11: Enumeration values for type TRegisterType

Value	Explanation
Address	Address register.
FP	Floating-point register.
Int	Integer register.
Invalid	Invalid register.
MMX	MMX register.
MultiMedia	Multimedia register.
Special	Special register.

`TRegisterType` describes the type of the CPU register.

```
TSetPropValue = procedure(Instance: TObject; PropInfo: PPropInfo;
  const Value: Variant)
```

The callback function must set the property with name `PropName` of instance `Instance` to `Value`.

```
TSetVariantProp = procedure(Instance: TObject; PropInfo: PPropInfo
  ;
                              const Value: Variant)
```

The callback function must set the variant property with name `PropName` of instance to `Value`.

```
TSubRegister = (None,Lo,Hi,Word,DWord,QWord,FloatSingle,FloatDouble
  ,
                FloatQuad,MultiMediaSingle,MultiMediaDouble,
  MultiMediaWhole,MultiMediaX,MultiMediaY)
```

Table 78.12: Enumeration values for type TSubRegister

Value	Explanation
DWord	DWord size.
FloatDouble	Double-sized float value.
FloatQuad	Quadruple-sized float value.
FloatSingle	Single-sized float value.
Hi	High.
Lo	Low.
MultiMediaDouble	Multimedia register double-sized float.
MultiMediaSingle	Multimedia register single-sized float.
MultiMediaWhole	Multimedia register whole-sized.
MultiMediaX	Multimedia register X position.
MultiMediaY	Multimedia register Y position.
None	None.
QWord	QWord size.
Word	Word size.

`TSubRegister` describes CPU sub-register sizes or locations.

```
TTypeInfo = record
public
  Kind : TTypeKind;
  Name : ShortString
  ;
end
```

The `TypeInfo` function returns a pointer to a `TTypeInfo` record.

Note that the Name field is stored with as much bytes as needed to store the name, it is not padded to 255 characters. The type data immediately follows the `TTypeInfo` record as a TTypeData (2091) record.

```
TTypeKind = System.TTypeKind
```

`TTypeKind` was moved to the system unit, see #rtl.system.TTypeKind (1429)

```
TTypeKinds = Set of TTypeKind
```

Set of TTypeKind (2028) enumeration.

```
TVmtFieldClassTab = packed record
public
  Count : Word;
  ClassRef
  : Array[0..0] of PClass;
end
```

`TVmtFieldClassTab` describes the fields of class type. A pointer to this table is present in TVmtFieldTable (2095)

```
TVmtMethodEntry = packed record
public
  Name : PShortString;
  CodeAddress
  : CodePointer;
end
```

`TVmtMethodEntry` is a description of the VMT info generated for a method of a `class`. This is different from interface method RTTI info, which is described by TIntfMethodEntry (2065)

78.7 Procedures and functions

78.7.1 AddEnumElementAliases

Synopsis: Define aliases for enumerated elements.

Declaration:
```
procedure AddEnumElementAliases(aTypeInfo: PTypeInfo;
                                const aNames: Array of string;
                                aStartValue: Integer)
```

Visibility: default

Description: `AddEnumElementAliases` defines aliases for enumerated values of type `aTypeInfo`. The aliases are specified in `aNames`, and are applied as of the element with ordinal value `aStartIndex`.

The aliases will be used in GetEnumValue (2034)

See also: RemoveEnumElementAliases (2052), GetEnumValue (2034), GetEnumeratedAliasValue (2032)

78.7.2 AlignPTypeInfo

Synopsis: Align `TTypeInfo` info.

Declaration:
```
function AlignPTypeInfo(p: Pointer) : Pointer
```

Visibility: default

Description: `AlignPTypeInfo` will align the pointer `p` on a correct boundary for TParamFlags (2026) data for the current platform and returns the resulting address.

See also: align (2016), AlignTypeData (2030), AlignPTypeInfo (2029)

78.7.3 AlignTParamFlags

Synopsis: Align `TParamFlags` info.

Declaration:
```
function AlignTParamFlags(p: Pointer) : Pointer
```

Visibility: default

Description: `AlignTParamFlags` will align the pointer `p` on a correct boundary for TParamFlags (2026) data for the current platform and returns the resulting address.

See also: align (2016), AlignTypeData (2030), AlignPTypeInfo (2029)

78.7.4 AlignTypeData

Synopsis: Align type data.

Declaration:
```
function AlignTypeData(p: Pointer) : Pointer
```

Visibility: default

Description: `AlignTypeData` will align the pointer `p` on a correct boundary for type data for the current platform and returns the resulting address.

See also: align (2016), AlignTParamFlags (2030), AlignPTypeInfo (2029)

78.7.5 DerefTypeInfoPtr

Synopsis: Dereference type info pointer.

Declaration:
```
function DerefTypeInfoPtr(Info: PPTypeInfo) : PTypeInfo
```

Visibility: default

Description: `DerefTypeInfoPtr` can be used to dereference the `Info` type info pointer if need be: in FPC version 3.0, this pointer does not need dereferencing, but in FPC version 3.1, the pointer must be dereferenced. This is useful for code that must work in version 3.0 and later.

Errors: The function handles nil values well.

See also: PTypeInfo (2023)

78.7.6 FindPropInfo

Synopsis: Return property information by property name.

Declaration:
```
function FindPropInfo(Instance: TObject; const PropName: string)
                      : PPropInfo
function FindPropInfo(Instance: TObject; const PropName: string;
                      AKinds: TTypeKinds) : PPropInfo
function FindPropInfo(AClass: TClass; const PropName: string)
                      : PPropInfo
function FindPropInfo(AClass: TClass; const PropName: string;
                      AKinds: TTypeKinds) : PPropInfo
```

Visibility: default

Description: `FindPropInfo` examines the published property information of a class and returns a pointer to the property information for property `PropName`. The class to be examined can be specified in one of two ways:

AClassa class pointer.

Instancean instance of the class to be investigated.

If the property does not exist, a `EPropertyError` exception will be raised. The GetPropInfo (2041) function has the same function as the `FindPropInfo` function, but returns `Nil` if the property does not exist.

Errors: Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetPropInfo (2041), GetPropList (2042), GetPropInfos (2041)

Listing: ./examples/typinfex/ex14.pp

```
Program example13;

{ This program demonstrates the FindPropInfo function }

{$mode objfpc}

uses
  rttiobj,typinfo,sysutils;


Var
  O : TMyTestObject;
  PT : PTypeData;
  PI : PPropInfo;
  I,J : Longint;
  PP : PPropList;
  prl : PPropInfo;

begin
  O:=TMyTestObject.Create;
  PI:=FindPropInfo(O,'BooleanField');
  Writeln('FindPropInfo(Instance,BooleanField) : ',PI^.Name);
  PI:=FindPropInfo(O.ClassType,'ByteField');
  Writeln('FindPropInfo(Class,ByteField)       : ',PI^.Name);
  Write  ('FindPropInfo(Class,NonExistingProp) : ');
  Try
    PI:=FindPropInfo(O,'NonExistingProp');
  except
    On E: Exception do
      Writeln('Caught exception "',E.ClassName,'" with message : ',E.Message);
  end;
  O.Free;
end.
```

78.7.7 GetDynArrayProp

Synopsis: Get the value of a dynamic array valued property.

Declaration:
```
function GetDynArrayProp(Instance: TObject; const PropName: string)
                         : Pointer
```

```
function GetDynArrayProp(Instance: TObject; PropInfo: PPropInfo)
                         : Pointer
```

Visibility: default

Description: `GetDynArrayProp` returns the value of a dynamic array valued property of the object `Instance` as a pointer. The property to read can be specified using the name `PropName` or the property info `PropInfo`.

See also: SetDynArrayProp (2052)

78.7.8 GetEnumeratedAliasValue

Synopsis: Return ordinal value for an enumerated value.

Declaration:
```
function GetEnumeratedAliasValue(aTypeInfo: PTypeInfo;
                                 const aName: string) : Integer
```

Visibility: default

Description: `GetEnumeratedAliasValue` will return the ordinal value of the alias `aName` for type `aTypeInfo`.

See also: AddEnumElementAliases (2029), RemoveEnumElementAliases (2052), GetEnumValue (2034)

78.7.9 GetEnumName

Synopsis: Return name of enumeration constant.

Declaration:
```
function GetEnumName(TypeInfo: PTypeInfo; Value: Integer) : string
```

Visibility: default

Description: `GetEnumName` scans the type information for the enumeration type described by `TypeInfo` and returns the name of the enumeration constant for the element with ordinal value equal to `Value`.

If `Value` is out of range, the first element of the enumeration type is returned. The result is returned in the case that was used in the declaration. (In earlier versions of FPC, the name was lowercased).

This can be used in combination with `GetOrdProp` to stream a property of an enumerated type.

Errors: No check is done to determine whether `TypeInfo` really points to the type information for an enumerated type.

See also: GetOrdProp (2040), GetEnumValue (2034)

Listing: ./examples/typinfex/ex9.pp

```
program example9;

{ This program demonstrates the GetEnumName, GetEnumValue functions }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  TI : PTypeInfo;

begin
```

```
  O:=TMyTestObject.Create;
  TI:=GetPropInfo(O,'MyEnumField')^.PropType;
  WriteIn('GetEnumName              : ',GetEnumName(TI,Ord(O.MyEnumField)));
  WriteIn('GetEnumValue(mefirst)  : ',GetEnumName(TI,GetEnumValue(TI,'mefirst')));
  O.Free;
end.
```

78.7.10 GetEnumNameCount

Synopsis: Return number of names in an enumerated type.

Declaration: `function GetEnumNameCount(enum1: PTypeInfo) : SizeInt`

Visibility: default

Description: `GetEnumNameCount` returns the number of values (names) in the enumerated type, described by `enum1`

Errors: No checking is done to see whether `Enum1` is really type information of an enumerated type.

See also: GetEnumValue (2034), GetEnumName (2032)

78.7.11 GetEnumProp

Synopsis: Return the value of an enumeration type property.

Declaration:
```
function GetEnumProp(Instance: TObject; const PropName: string) : string
function GetEnumProp(Instance: TObject; const PropInfo: PPropInfo)
                        : string
```

Visibility: default

Description: `GetEnumProp` returns the value of an property of an enumerated type and returns the name of the enumerated value for the object `Instance`. The property whose value must be returned can be specified by its property info in `PropInfo` or by its name in `PropName`

Errors: No check is done to determine whether `PropInfo` really points to the property information for an enumerated type. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetEnumProp (2053), GetOrdProp (2040), GetStrProp (2046), GetInt64Prop (2035), GetMethodProp (2036), GetSetProp (2044), GetObjectProp (2038), GetEnumProp (2033)

Listing: ./examples/typinfex/ex2.pp

```
program example2;

{ This program demonstrates the GetEnumProp function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;
  TI : PTypeInfo;
```

```
begin
  O:=TMyTestObject.Create;
  PI:=GetPropInfo(O,'MyEnumField');
  TI:=PI^.PropType;
  Writeln('Enum property     : ');
  Writeln('Value                    : ',GetEnumName(TI,Ord(O.MyEnumField)));
  Writeln('Get (name)               : ',GetEnumProp(O,'MyEnumField'));
  Writeln('Get (propinfo)           : ',GetEnumProp(O,PI));
  SetEnumProp(O,'MyEnumField','meFirst');
  Writeln('Set (name,meFirst)       : ',GetEnumName(TI,Ord(O.MyEnumField)));
  SetEnumProp(O,PI,'meSecond');
  Writeln('Set (propinfo,meSecond)  : ',GetEnumName(TI,Ord(O.MyEnumField)));
  O.Free;
end.
```

78.7.12 GetEnumValue

Synopsis: Get ordinal value for enumerated type by name.

Declaration: `function GetEnumValue(TypeInfo: PTypeInfo; const Name: string) : Integer`

Visibility: default

Description: `GetEnumValue` scans the type information for the enumeration type described by `TypeInfo` and returns the ordinal value for the element in the enumerated type that has identifier `Name`. The identifier is searched in a case-insensitive manner.

This can be used to set the value of enumerated properties from a stream.

For an example, see GetEnumName (2032).

Errors: If `Name` is not found in the list of enumerated values, then -1 is returned. No check is done whether `TypeInfo` points to the type information for an enumerated type.

See also: GetEnumName (2032), SetOrdProp (2055)

78.7.13 GetFloatProp

Synopsis: Return value of floating point property.

Declaration:
```
function GetFloatProp(Instance: TObject; PropInfo: PPropInfo) : Extended
function GetFloatProp(Instance: TObject; const PropName: string)
                      : Extended
```

Visibility: default

Description: `GetFloatProp` returns the value of the float property described by `PropInfo` or with name `Propname` for the object `Instance`. All float types are converted to extended.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid float property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetFloatProp (2053), GetOrdProp (2040), GetStrProp (2046), GetInt64Prop (2035), GetMethodProp (2036), GetSetProp (2044), GetObjectProp (2038), GetEnumProp (2033)

Listing: ./examples/typinfex/ex4.pp

```
program example4;

{ This program demonstrates the GetFloatProp function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  Writeln('Real property : ');
  PI:=GetPropInfo(O,'RealField');
  Writeln('Value              : ',O.RealField);
  Writeln('Get (name)         : ',GetFloatProp(O,'RealField'));
  Writeln('Get (propinfo)     : ',GetFloatProp(O,PI));
  SetFloatProp(O,'RealField',system.Pi);
  Writeln('Set (name,pi)      : ',O.RealField);
  SetFloatProp(O,PI,exp(1));
  Writeln('Set (propinfo,e)   : ',O.RealField);
  Writeln('Extended property : ');
  PI:=GetPropInfo(O,'ExtendedField');
  Writeln('Value              : ',O.ExtendedField);
  Writeln('Get (name)         : ',GetFloatProp(O,'ExtendedField'));
  Writeln('Get (propinfo)     : ',GetFloatProp(O,PI));
  SetFloatProp(O,'ExtendedField',system.Pi);
  Writeln('Set (name,pi)      : ',O.ExtendedField);
  SetFloatProp(O,PI,exp(1));
  Writeln('Set (propinfo,e)   : ',O.ExtendedField);
  O.Free;
end.
```

78.7.14 GetInt64Prop

Synopsis: return value of an Int64 property.

Declaration:
```
function GetInt64Prop(Instance: TObject; PropInfo: PPropInfo) : Int64
function GetInt64Prop(Instance: TObject; const PropName: string) : Int64
```

Visibility: default

Description: Publishing of Int64 properties is not yet supported by Free Pascal. This function is provided for Delphi compatibility only at the moment.

`GetInt64Prop` returns the value of the property of type `Int64` that is described by `PropInfo` or with name `Propname` for the object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid `Int64` property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception

See also: SetInt64Prop (2053), GetOrdProp (2040), GetStrProp (2046), GetFloatProp (2034), GetMethodProp (2036), GetSetProp (2044), GetObjectProp (2038), GetEnumProp (2033)

Listing: ./examples/typinfex/ex15.pp

```
program example15;

{ This program demonstrates the GetInt64Prop function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  Writeln('Int64 property : ');
  PI:=GetPropInfo(O,'Int64Field');
  Writeln('Value              : ',O.Int64Field);
  Writeln('Get (name)         : ',GetInt64Prop(O,'Int64Field'));
  Writeln('Get (propinfo)     : ',GetInt64Prop(O,PI));
  SetInt64Prop(O,'Int64Field',12345);
  Writeln('Set (name,12345)     : ',O.Int64Field);
  SetInt64Prop(O,PI,54321);
  Writeln('Set (propinfo,54321) : ',O.Int64Field);
  O.Free;
end.
```

78.7.15 GetInterfaceProp

Synopsis: Return interface-typed property.

Declaration:
```
function GetInterfaceProp(Instance: TObject; const PropName: string)
                          : IInterface
function GetInterfaceProp(Instance: TObject; PropInfo: PPropInfo)
                          : IInterface
```

Visibility: default

Description: `GetInterfaceProp` returns the interface which the property described by `PropInfo` or with name `Propname` points to for object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid method property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetInterfaceProp (2054), GetOrdProp (2040), GetStrProp (2046), GetFloatProp (2034), GetInt64Prop (2035), GetSetProp (2044), GetObjectProp (2038), GetEnumProp (2033)

78.7.16 GetMethodProp

Synopsis: Return value of a method property.

Declaration:
```
function GetMethodProp(Instance: TObject; PropInfo: PPropInfo) : TMethod
function GetMethodProp(Instance: TObject; const PropName: string)
                       : TMethod
```

Visibility: default

Description: `GetMethodProp` returns the method the property described by `PropInfo` or with name `Propname` for object `Instance`. The return type `TMethod` is defined in the SysUtils unit as:

```
TMethod = packed record
  Code, Data: Pointer;
end;
```

`Data` points to the instance of the class with the method `Code`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid method property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetMethodProp (2054), GetOrdProp (2040), GetStrProp (2046), GetFloatProp (2034), GetInt64Prop (2035), GetSetProp (2044), GetObjectProp (2038), GetEnumProp (2033)

Listing: ./examples/typinfex/ex6.pp

```
program example6;

{ This program demonstrates the GetMethodProp function }

{$mode objfpc}

uses rttiobj,typinfo,sysutils;

Type
  TNotifyObject = Class(TObject)
    Procedure Notification1(Sender : TObject);
    Procedure Notification2(Sender : TObject);
  end;

Procedure TNotifyObject.Notification1(Sender : TObject);

begin
  Write('Received notification 1 of object with class: ');
  Writeln(Sender.ClassName);
end;

Procedure TNotifyObject.Notification2(Sender : TObject);

begin
  Write('Received notification 2 of object with class: ');
  Writeln(Sender.ClassName);
end;

Var
  O : TMyTestObject;
  PI : PPropInfo;
  NO : TNotifyObject;
  M : TMethod;

Procedure PrintMethod (Const M : TMethod);

begin
  If (M.Data=Pointer(NO)) Then
```

```
    If (M.Code=Pointer(@TNotifyObject.Notification1)) then
      WriteIn('Notification1')
    else If (M.Code=Pointer(@TNotifyObject.Notification2)) then
      WriteIn('Notification2')
    else
      begin
      Write('Unknown method adress (data:');
      Write(hexStr(Longint(M.data),8));
      WriteIn(',code:',hexstr(Longint(M.Code),8),')');
      end;
end;


begin
  O:=TMyTestObject.Create;
  NO:=TNotifyObject.Create;
  O.NotifyEvent:=@NO.Notification1;
  PI:=GetPropInfo(O,'NotifyEvent');
  WriteIn('Method property : ');
  Write('Notifying                    : ');
  O.Notify;
  Write('Get (name)                   : ');
  M:=GetMethodProp(O,'NotifyEvent');
  PrintMethod(M);
  Write('Notifying                    : ');
  O.Notify;
  Write('Get (propinfo)               : ');
  M:=GetMethodProp(O,PI);
  PrintMethod(M);
  M:=TMethod(@NO.Notification2);
  SetMethodProp(O,'NotifyEvent',M);
  Write('Set (name,Notification2)     : ');
  M:=GetMethodProp(O,PI);
  PrintMethod(M);
  Write('Notifying                    : ');
  O.Notify;
  Write('Set (propinfo,Notification1) : ');
  M:=TMethod(@NO.Notification1);
  SetMethodProp(O,PI,M);
  M:=GetMethodProp(O,PI);
  PrintMethod(M);
  Write('Notifying                    : ');
  O.Notify;
  O.Free;
end.
```

78.7.17 GetObjectProp

Synopsis: Return value of an object-type property.

Declaration:
```
function GetObjectProp(Instance: TObject; const PropName: string)
                       : TObject
function GetObjectProp(Instance: TObject; const PropName: string;
                       MinClass: TClass) : TObject
function GetObjectProp(Instance: TObject; PropInfo: PPropInfo) : TObject
function GetObjectProp(Instance: TObject; PropInfo: PPropInfo;
                       MinClass: TClass) : TObject
```

Visibility: default

Description: `GetObjectProp` returns the object which the property described by `PropInfo` with name `Propname` points to for object `Instance`.

If `MinClass` is specified, then if the object is not descendent of class `MinClass`, then `Nil` is returned.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid method property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetMethodProp (2054), GetOrdProp (2040), GetStrProp (2046), GetFloatProp (2034), GetInt64Prop (2035), GetSetProp (2044), GetObjectProp (2038), GetEnumProp (2033)

Listing: ./examples/typinfex/ex5.pp

```
program example5;

{ This program demonstrates the GetObjectProp function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;
  NO1,NO2 : TNamedObject;

begin
  O:=TMyTestObject.Create;
  NO1:=TNamedObject.Create;
  NO1.ObjectName:='First named object';
  NO2:=TNamedObject.Create;
  NO2.ObjectName:='Second named object';
  O.ObjField:=NO1;
  Writeln('Object property : ');
  PI:=GetPropInfo(O,'ObjField');
  Write('Property class      : ');
  Writeln(GetObjectPropClass(O,'ObjField').ClassName);
  Write('Value               : ');
  Writeln((O.ObjField as TNamedObject).ObjectName);
  Write('Get (name)          : ');
  Writeln((GetObjectProp(O,'ObjField') As TNamedObject).ObjectName);
  Write('Get (propinfo)      : ');
  Writeln((GetObjectProp(O,PI,TObject) as TNamedObject).ObjectName);
  SetObjectProp(O,'ObjField',NO2);
  Write('Set (name,NO2)      : ');
  Writeln((O.ObjField as TNamedObject).ObjectName);
  SetObjectProp(O,PI,NO1);
  Write('Set (propinfo,NO1)  : ');
  Writeln((O.ObjField as TNamedObject).ObjectName);
  O.Free;
end.
```

78.7.18 GetObjectPropClass

Synopsis: Return class of property.

Declaration:
```
function GetObjectPropClass(Instance: TObject; const PropName: string)
                                  : TClass
function GetObjectPropClass(AClass: TClass; const PropName: string)
                                  : TClass
```

Visibility: default

Description: `GetObjectPropClass` returns the declared class of the property with name `PropName`. This may not be the actual class of the property value.

For an example, see GetObjectProp (2038).

Errors: No checking is done whether `Instance` is non-nil. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetMethodProp (2054), GetOrdProp (2040), GetStrProp (2046), GetFloatProp (2034), GetInt64Prop (2035)

78.7.19 GetOrdProp

Synopsis: Get the value of an ordinal property.

Declaration:
```
function GetOrdProp(Instance: TObject; PropInfo: PPropInfo) : Int64
function GetOrdProp(Instance: TObject; const PropName: string) : Int64
```

Visibility: default

Description: `GetOrdProp` returns the value of the ordinal property described by `PropInfo` or with name `PropName` for the object `Instance`. The value is returned as a longint, which should be typecast to the needed type.

Ordinal properties that can be retrieved include:

Integers and subranges of integers The value of the integer will be returned.

Enumerated types and subranges of enumerated types The ordinal value of the enumerated type will be returned.

SetsIf the base type of the set has less than 31 possible values. If a bit is set in the return value, then the corresponding element of the base ordinal class of the set type must be included in the set.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid ordinal property of `Instance` Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetOrdProp (2055), GetStrProp (2046), GetFloatProp (2034), GetInt64Prop (2035), GetMethodProp (2036), GetSetProp (2044), GetObjectProp (2038), GetEnumProp (2033)

Listing: ./examples/typinfex/ex1.pp

```
program example1;

{ This program demonstrates the GetOrdProp function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
```

```
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  Writeln('Boolean property     : ');
  Writeln('Value                : ',O.BooleanField);
  Writeln('Ord(Value)           : ',Ord(O.BooleanField));
  Writeln('Get (name)           : ',GetOrdProp(O,'BooleanField'));
  PI:=GetPropInfo(O,'BooleanField');
  Writeln('Get (propinfo)       : ',GetOrdProp(O,PI));
  SetOrdProp(O,'BooleanField',Ord(False));
  Writeln('Set (name,false)     : ',O.BooleanField);
  SetOrdProp(O,PI,Ord(True));
  Writeln('Set (propinfo,true)  : ',O.BooleanField);
  O.Free;
end.
```

78.7.20 GetPropInfo

Synopsis: Return property type information, by property name.

Declaration:
```
function GetPropInfo(TypeInfo: PTypeInfo; const PropName: string)
                     : PPropInfo
function GetPropInfo(TypeInfo: PTypeInfo; const PropName: string;
                     AKinds: TTypeKinds) : PPropInfo
function GetPropInfo(Instance: TObject; const PropName: string)
                     : PPropInfo
function GetPropInfo(Instance: TObject; const PropName: string;
                     AKinds: TTypeKinds) : PPropInfo
function GetPropInfo(AClass: TClass; const PropName: string) : PPropInfo
function GetPropInfo(AClass: TClass; const PropName: string;
                     AKinds: TTypeKinds) : PPropInfo
```

Visibility: default

Description: `GetPropInfo` returns a pointer to the `TPropInfo` record for the `PropName` property of a class. The class to examine can be specified in one of three ways:

InstanceAn instance of the class.

AClassA class pointer to the class.

TypeInfoA pointer to the type information of the class.

In each of these three ways, if `AKinds` is specified, if the property has `TypeKind` which is not included in `Akinds`, `Nil` will be returned.

For an example, see most of the other functions.

Errors: If the property `PropName` does not exist, `Nil` is returned.

See also: GetPropInfos (2041), GetPropList (2042)

78.7.21 GetPropInfos

Synopsis: Return a list of published properties.

Declaration:
```
procedure GetPropInfos(TypeInfo: PTypeInfo; PropList: PPropList)
```

Visibility: default

Description: `GetPropInfos` stores pointers to the property information of all published properties of a class with class info `TypeInfo` in the list pointed to by `Proplist`. The `PropList` pointer must point to a memory location that contains enough space to hold all properties of the class and its parent classes.

Errors: No checks are done to see whether `PropList` points to a memory area that is big enough to hold all pointers.

See also: GetPropInfo (2041), GetPropList (2042)

Listing: ./examples/typinfex/ex12.pp

```
Program example12;

{ This program demonstrates the GetPropInfos function }

uses
  rttiobj,typinfo;


Var
  O : TMyTestObject;
  PT : PTypeData;
  PI : PTypeInfo;
  I,J : Longint;
  PP : PPropList;
  prI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  PI:=O.ClassInfo;
  PT:=GetTypeData(PI);
  Writeln('Property Count : ',PT^.PropCount);
  GetMem (PP,PT^.PropCount*SizeOf(Pointer));
  GetPropInfos(PI,PP);
  For I:=0 to PT^.PropCount-1 do
    begin
    With PP^[i]^ do
      begin
      Write('Property ',i+1:3,': ',name:30);
      writeln('  Type: ',TypeNames[typinfo.PropType(O,Name)]);
      end;
    end;
  FreeMem(PP);
  O.Free;
end.
```

78.7.22 GetPropList

Synopsis: Return a list of a certain type of published properties.

Declaration:
```
function GetPropList(TypeInfo: PTypeInfo; TypeKinds: TTypeKinds;
                     PropList: PPropList; Sorted: Boolean) : LongInt
function GetPropList(TypeInfo: PTypeInfo; out PropList: PPropList)
                     : SizeInt
```

```
function GetPropList(AClass: TClass; out PropList: PPropList) : Integer
function GetPropList(Instance: TObject; out PropList: PPropList)
                     : Integer
```

Visibility: default

Description: `GetPropList` stores pointers to property information of the class with class info `TypeInfo` for properties of kind `TypeKinds` in the list pointed to by `Proplist`. `PropList` must contain enough space to hold all properties.

The function returns the number of pointers that matched the criteria and were stored in `PropList`.

Errors: No checks are done to see whether `PropList` points to a memory area that is big enough to hold all pointers.

See also: GetPropInfos (2041), GetPropInfo (2041)

Listing: ./examples/typinfex/ex13.pp

```
Program example13;

{ This program demonstrates the GetPropList function }

uses
  rttiobj,typinfo;


Var
  O : TMyTestObject;
  PT : PTypeData;
  PI : PTypeInfo;
  I,J : Longint;
  PP : PPropList;
  prI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  PI:=O.ClassInfo;
  PT:=GetTypeData(PI);
  Writeln('Total property Count : ',PT^.PropCount);
  GetMem (PP,PT^.PropCount*SizeOf(Pointer));
  J:=GetPropList(PI,OrdinalTypes,PP);
  Writeln('Ordinal property Count : ',J);
  For I:=0 to J-1 do
    begin
    With PP^[i]^ do
      begin
      Write('Property ',i+1:3,': ',name:30);
      writeln('  Type: ',TypeNames[typinfo.PropType(O,Name)]);
      end;
    end;
  FreeMem(PP);
  O.Free;
end.
```

78.7.23 GetPropValue

Synopsis: Get property value as a string.

Declaration:
```
function GetPropValue(Instance: TObject; const PropName: string)
                          : Variant
function GetPropValue(Instance: TObject; const PropName: string;
                      PreferStrings: Boolean) : Variant
function GetPropValue(Instance: TObject; PropInfo: PPropInfo) : Variant
function GetPropValue(Instance: TObject; PropInfo: PPropInfo;
                      PreferStrings: Boolean) : Variant
```

Visibility: default

Description: Due to missing `Variant` support, `GetPropValue` is not yet implemented. The declaration is provided for compatibility with Delphi.

78.7.24 GetRawByteStrProp

Synopsis: Return the value of a string property as a `RawByteString`.

Declaration:
```
function GetRawbyteStrProp(Instance: TObject; PropInfo: PPropInfo)
                           : RawByteString
function GetRawByteStrProp(Instance: TObject; const PropName: string)
                           : RawByteString
```

Visibility: default

Description: `GetRawbyteStrProp` returns the value of a string property of the object `Instance` as a `RawByteString`. This means it has the codepage of the actual value, no codepage translation is done. The property to read can be specified using the name `PropName` or the property info `PropInfo`.

See also: GetStrProp (2046)

78.7.25 GetRawInterfaceProp

Synopsis: Get a raw (CORBA) interface property.

Declaration:
```
function GetRawInterfaceProp(Instance: TObject; const PropName: string)
                             : Pointer
function GetRawInterfaceProp(Instance: TObject; PropInfo: PPropInfo)
                             : Pointer
```

Visibility: default

Description: `GetRawInterfaceProp` can be used to retrieve the value of a published CORBA interface property with name `PropName` from object `Instance`. Alternatively, the required property information can be specified by `PropInfo` instead of the property name. In difference with the GetInterfaceProp (2036) function, no reference counting is done.

Errors: If the property `PropName` does not exist, an `EPropertyError` exception is raised.

See also: GetInterfaceProp (2036), SetRawInterfaceProp (2056)

78.7.26 GetSetProp

Synopsis: Return the value of a set property.

Declaration:
```
function GetSetProp(Instance: TObject; const PropName: string) : string
function GetSetProp(Instance: TObject; const PropName: string;
                    Brackets: Boolean) : string
function GetSetProp(Instance: TObject; const PropInfo: PPropInfo;
                    Brackets: Boolean) : string
```

Visibility: default

Description: `GetSetProp` returns the contents of a set property as a string. The property to be returned can be specified by it's name in `PropName` or by its property information in `PropInfo`.

The returned set is a string representation of the elements in the set as returned by SetToString (2057). The `Brackets` option can be used to enclose the string representation in square brackets.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid ordinal property of `Instance` Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetSetProp (2057), GetStrProp (2046), GetFloatProp (2034), GetInt64Prop (2035), GetMethodProp (2036)

Listing: ./examples/typinfex/ex7.pp

```
program example7;

{ This program demonstrates the GetSetProp function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

Function SetAsString (ASet : TMyEnums) : String;

Var
  i : TmyEnum;

begin
  result:='';
  For i:=mefirst to methird do
    If i in ASet then
      begin
      If (Result<>'') then
        Result:=Result+',';
      Result:=Result+MyEnumNames[i];
      end;
end;

Var
  S : TMyEnums;

begin
  O:=TMyTestObject.Create;
  O.SetField:=[mefirst,meSecond,meThird];
  Writeln('Set property     : ');
  Writeln('Value                          : ',SetAsString(O.SetField));
```

```
  Writeln('Ord(Value)                    : ',Byte(O.SetField));
  Writeln('Get (name)                    : ',GetSetProp(O,'SetField'));
  PI:=GetPropInfo(O,'SetField');
  Writeln('Get (propinfo)                : ',GetSetProp(O,PI,false));
  S:=[meFirst,meThird];
  SetOrdProp(O,'SetField',Byte(S));
  Write('Set (name,[mefirst,methird]) : ');
  Writeln(SetAsString(O.SetField));
  S:=[meSecond];
  SetOrdProp(O,PI,Byte(S));
  Write('Set (propinfo,[meSecond])    : ');
  Writeln(SetAsString(O.SetField));
  O.Free;
end.
```

78.7.27 GetStrProp

Synopsis: Return the value of a string property.

Declaration:
```
function GetStrProp(Instance: TObject; PropInfo: PPropInfo) : Ansistring
function GetStrProp(Instance: TObject; const PropName: string) : string
```

Visibility: default

Description: `GetStrProp` returns the value of the string property described by `PropInfo` or with name `PropName` for object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid string property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: SetStrProp (2057), SetWideStrProp (2059), GetOrdProp (2040), GetFloatProp (2034), GetInt64Prop (2035), GetMethodProp (2036)

Listing: ./examples/typinfex/ex3.pp

```
program example3;

{ This program demonstrates the GetStrProp function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  PI:=GetPropInfo(O,'AnsiStringField');
  Writeln('String property : ');
  Writeln('Value                    : ',O.AnsiStringField);
  Writeln('Get (name)               : ',GetStrProp(O,'AnsiStringField'));
  Writeln('Get (propinfo)           : ',GetStrProp(O,PI));
  SetStrProp(O,'AnsiStringField','First');
  Writeln('Set (name,''First'')       : ',O.AnsiStringField);
```

```
  SetStrProp(O,PI,'Second');
  Writeln('Set (propinfo,''Second'') : ',O.AnsiStringField);
  O.Free;
end.
```

78.7.28 GetTypeData

Synopsis: Return a pointer to type data, based on type information.

Declaration:
```
function GetTypeData(TypeInfo: PTypeInfo) : PTypeData
```

Visibility: default

Description: `GetTypeData` returns a pointer to the `TTypeData` record that follows after the `TTypeInfo` record pointed to by `TypeInfo`. It essentially skips the `Kind` and `Name` fields in the `TTypeInfo` record.

Errors: None.

78.7.29 GetUnicodeStrProp

Synopsis: Get UnicodeString-valued property.

Declaration:
```
function GetUnicodeStrProp(Instance: TObject; PropInfo: PPropInfo)
                           : UnicodeString
function GetUnicodeStrProp(Instance: TObject; const PropName: string)
                           : UnicodeString
```

Visibility: default

Description: `GetUnicodeStrProp` returns the UnicodeString property from `Instance`, where the property is identified by the `PropInfo` pointer or the `PropertyName`.

Errors: If no property of the indicated name exists, or the value is not a Unicode string, an exception will occur.

See also: GetStrProp (2046), SetUnicodeStrProp (2058)

78.7.30 GetVariantProp

Synopsis: Return the value of a variant property.

Declaration:
```
function GetVariantProp(Instance: TObject; PropInfo: PPropInfo)
                        : Variant
function GetVariantProp(Instance: TObject; const PropName: string)
                        : Variant
```

Visibility: default

Description: Due to missing Variant support, the `GetVariantProp` function is not yet implemented. Provided for Delphi compatibility only.

See also: SetVariantProp (2059)

78.7.31 GetWideStrProp

Synopsis: Read a widestring property.

Declaration:
```
function GetWideStrProp(Instance: TObject; PropInfo: PPropInfo)
                        : WideString
function GetWideStrProp(Instance: TObject; const PropName: string)
                        : WideString
```

Visibility: default

Description: `GetWideStrProp` returns the value of the widestring property described by `PropInfo` or with name `PropName` for object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid widestring property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetStrProp (2046), SetWideStrProp (2059), GetOrdProp (2040), GetFloatProp (2034), GetInt64Prop (2035), GetMethodProp (2036)

78.7.32 IsPublishedProp

Synopsis: Check whether a published property exists.

Declaration:
```
function IsPublishedProp(Instance: TObject; const PropName: string)
                         : Boolean
function IsPublishedProp(AClass: TClass; const PropName: string)
                         : Boolean
```

Visibility: default

Description: `IsPublishedProp` returns true if a class has a published property with name `PropName`. The class can be specified in one of two ways:

AClass A class pointer to the class.

Instance An instance of the class.

Errors: No checks are done to ensure `Instance` or `AClass` are valid pointers. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: IsStoredProp (2049), PropIsType (2050)

Listing: ./examples/typinfex/ex10.pp

```
program example10;

{ This program demonstrates the IsPublishedProp function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
```

```
  O:=TMyTestObject.Create;
  Writeln('Property tests     : ');
  Write('IsPublishedProp(O,BooleanField)      : ');
  Writeln(IsPublishedProp(O,'BooleanField'));
  Write('IsPublishedProp(Class,BooleanField) : ');
  Writeln(IsPublishedProp(O.ClassType,'BooleanField'));
  Write('IsPublishedProp(O,SomeField)         : ');
  Writeln(IsPublishedProp(O,'SomeField'));
  Write('IsPublishedProp(Class,SomeField)     : ');
  Writeln(IsPublishedProp(O.ClassType,'SomeField'));
  O.Free;
end.
```

78.7.33 IsReadableProp

Synopsis: Can the property be read.

Declaration:
```
function IsReadableProp(PropInfo: PPropInfo) : Boolean
function IsReadableProp(Instance: TObject; const PropName: string)
                        : Boolean
function IsReadableProp(AClass: TClass; const PropName: string)
                        : Boolean
```

Visibility: default

Description: `IsReadableProp` returns `True` if the property described by information `PropInfo` can be read, i.e. has a `read` specifier. It returns `False` if the property can only be written.

See also: IsWriteableProp (2050)

78.7.34 IsStoredProp

Synopsis: Check whether a property is stored.

Declaration:
```
function IsStoredProp(Instance: TObject; PropInfo: PPropInfo) : Boolean
function IsStoredProp(Instance: TObject; const PropName: string)
                      : Boolean
```

Visibility: default

Description: `IsStoredProp` returns `True` if the `Stored` modifier evaluates to `True` for the property described by `PropInfo` or with name `PropName` for object `Instance`. It returns `False` otherwise. If the function returns `True`, this indicates that the property should be written when streaming the object `Instance`.

If there was no `stored` modifier in the declaration of the property, `True` will be returned.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: IsPublishedProp (2048), PropIsType (2050)

Listing: ./examples/typinfex/ex11.pp

```
program example11;

{ This program demonstrates the IsStoredProp function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  Writeln('Stored tests     : ');
  Write('IsStoredProp(O,StoredIntegerConstFalse)     : ');
  Writeln(IsStoredProp(O,'StoredIntegerConstFalse'));
  Write('IsStoredProp(O,StoredIntegerConstTrue)      : ');
  Writeln(IsStoredProp(O,'StoredIntegerConstTrue'));
  Write('IsStoredProp(O,StoredIntegerMethod)         : ');
  Writeln(IsStoredProp(O,'StoredIntegerMethod'));
  Write('IsStoredProp(O,StoredIntegerVirtualMethod) : ');
  Writeln(IsStoredProp(O,'StoredIntegerVirtualMethod'));
  O.Free;
end.
```

78.7.35 IsWriteableProp

Synopsis: Can the property be written.

Declaration:
```
function IsWriteableProp(PropInfo: PPropInfo) : Boolean
function IsWriteableProp(Instance: TObject; const PropName: string)
                         : Boolean
function IsWriteableProp(AClass: TClass; const PropName: string)
                         : Boolean
```

Visibility: default

Description: `IsWriteableProp` returns `True` if the property described by information `PropInfo` can be written to, i.e. has a `write` specifier. It returns `False` if the property can only be read.

See also: IsReadableProp (2049)

78.7.36 PropIsType

Synopsis: Check the type of a published property.

Declaration:
```
function PropIsType(Instance: TObject; const PropName: string;
                    TypeKind: TTypeKind) : Boolean
function PropIsType(AClass: TClass; const PropName: string;
                    TypeKind: TTypeKind) : Boolean
```

Visibility: default

Description: `PropIsType` returns `True` if the property with name `PropName` has type `TypeKind`. It returns `False` otherwise. The class to be examined can be specified in one of two ways:

AClassA class pointer.

InstanceAn instance of the class.

Errors: No checks are done to ensure `Instance` or `AClass` are valid pointers.Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: IsPublishedProp (2048), IsStoredProp (2049), PropType (2051)

Listing: ./examples/typinfex/ex16.pp

```
program example16;

{ This program demonstrates the PropIsType function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;

begin
  O:=TMyTestObject.Create;
  Writeln('Property tests     : ');
  Write('PropIsType(O,BooleanField,tkBool)      : ');
  Writeln(PropIsType(O,'BooleanField',tkBool));
  Write('PropIsType(Class,BooleanField,tkBool) : ');
  Writeln(PropIsType(O.ClassType,'BooleanField',tkBool));
  Write('PropIsType(O,ByteField,tkString)       : ');
  Writeln(PropisType(O,'ByteField',tkString));
  Write('PropIsType(Class,ByteField,tkString)   : ');
  Writeln(PropIsType(O.ClassType,'ByteField',tkString));
  O.Free;
end.
```

78.7.37 PropType

Synopsis: Return the type of a property.

Declaration:
```
function PropType(Instance: TObject; const PropName: string) : TTypeKind
function PropType(AClass: TClass; const PropName: string) : TTypeKind
```

Visibility: default

Description: `Proptype` returns the type of the property `PropName` for a class. The class to be examined can be specified in one of 2 ways:

AClassA class pointer.

InstanceAn instance of the class.

Errors: No checks are done to ensure `Instance` or `AClass` are valid pointers. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: IsPublishedProp (2048), IsStoredProp (2049), PropIsType (2050)

Listing: ./examples/typinfex/ex17.pp

```
program example17;

{ This program demonstrates the PropType function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;

begin
  O:=TMyTestObject.Create;
  Writeln('Property tests     : ');
  Write('PropType(O,BooleanField)      : ');
  Writeln(TypeNames[PropType(O,'BooleanField')]);
  Write('PropType(Class,BooleanField) : ');
  Writeln(TypeNames[PropType(O.ClassType,'BooleanField')]);
  Write('PropType(O,ByteField)         : ');
  Writeln(TypeNames[PropType(O,'ByteField')]);
  Write('PropType(Class,ByteField)     : ');
  Writeln(TypeNames[PropType(O.ClassType,'ByteField')]);
  O.Free;
end.
```

78.7.38 RemoveEnumElementAliases

Synopsis: Remove aliases for enumerated elements.

Declaration:
```
procedure RemoveEnumElementAliases(aTypeInfo: PTypeInfo)
```

Visibility: default

Description: `RemoveEnumElementAliases` removes all aliases for enumerated type `aTypeInfo`. The aliases must have been defined using AddEnumElementAliases (2029).

Errors: AddEnumElementAliases (2029)GetEnumValue (2034)GetEnumeratedAliasValue (2032)

78.7.39 SetDynArrayProp

Synopsis: SetSize the value of a dynamic array valued property.

Declaration:
```
procedure SetDynArrayProp(Instance: TObject; const PropName: string;
                          const Value: Pointer)
procedure SetDynArrayProp(Instance: TObject; PropInfo: PPropInfo;
                          const Value: Pointer)
```

Visibility: default

Description: `GetDynArrayProp` returns the value of a dynamic array valued property of the object `Instance` to the pointer `Value`. The property to read can be specified using the name `PropName` or the property info `PropInfo`.

See also: GetDynArrayProp (2031)

78.7.40 SetEnumProp

Synopsis: Set value of an enumerated-type property.

Declaration:
```
procedure SetEnumProp(Instance: TObject; const PropName: string;
                      const Value: string)
procedure SetEnumProp(Instance: TObject; const PropInfo: PPropInfo;
                      const Value: string)
```

Visibility: default

Description: `SetEnumProp` sets the property described by `PropInfo` or with name `PropName` to `Value`. `Value` must be a string with the name of the enumerate value, i.e. it can be used as an argument to GetEnumValue (2034).

For an example, see GetEnumProp (2033).

Errors: No checks are done to ensure `Instance` or `PropInfo` are valid pointers. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetEnumProp (2033), SetStrProp (2057), SetFloatProp (2053), SetInt64Prop (2053), SetMethodProp (2054)

78.7.41 SetFloatProp

Synopsis: Set value of a float property.

Declaration:
```
procedure SetFloatProp(Instance: TObject; const PropName: string;
                       Value: Extended)
procedure SetFloatProp(Instance: TObject; PropInfo: PPropInfo;
                       Value: Extended)
```

Visibility: default

Description: `SetFloatProp` assigns `Value` to the property described by `PropInfo` or with name `Propname` for the object `Instance`.

For an example, see GetFloatProp (2034).

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid float property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetFloatProp (2034), SetOrdProp (2055), SetStrProp (2057), SetInt64Prop (2053), SetMethodProp (2054)

78.7.42 SetInt64Prop

Synopsis: Set value of a Int64 property.

Declaration:
```
procedure SetInt64Prop(Instance: TObject; PropInfo: PPropInfo;
                       const Value: Int64)
procedure SetInt64Prop(Instance: TObject; const PropName: string;
                       const Value: Int64)
```

Visibility: default

Description: `SetInt64Prop` assigns `Value` to the property of type `Int64` that is described by `PropInfo` or with name `Propname` for the object `Instance`.

For an example, see GetInt64Prop (2035).

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid `Int64` property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetInt64Prop (2035), GetMethodProp (2036), SetOrdProp (2055), SetStrProp (2057), SetFloatProp (2053)

78.7.43 SetInterfaceProp

Synopsis: Set interface-valued property.

Declaration:
```
procedure SetInterfaceProp(Instance: TObject; const PropName: string;
                           const Value: IInterface)
procedure SetInterfaceProp(Instance: TObject; PropInfo: PPropInfo;
                           const Value: IInterface)
```

Visibility: default

Description: `SetInterfaceProp` assigns `Value` to the object property described by `PropInfo` or with name `Propname` for the object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid interface property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetInterfaceProp (2036), SetObjectProp (2055), SetOrdProp (2055), SetStrProp (2057), SetFloatProp (2053), SetInt64Prop (2053), SetMethodProp (2054)

78.7.44 SetMethodProp

Synopsis: Set the value of a method property.

Declaration:
```
procedure SetMethodProp(Instance: TObject; PropInfo: PPropInfo;
                        const Value: TMethod)
procedure SetMethodProp(Instance: TObject; const PropName: string;
                        const Value: TMethod)
```

Visibility: default

Description: `SetMethodProp` assigns `Value` to the method the property described by `PropInfo` or with name `Propname` for object `Instance`.

The type `TMethod` of the `Value` parameter is defined in the SysUtils unit as:

```
TMethod = packed record
  Code, Data: Pointer;
end;
```

`Data` should point to the instance of the class with the method `Code`.

For an example, see GetMethodProp (2036).

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid method property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetMethodProp (2036), SetOrdProp (2055), SetStrProp (2057), SetFloatProp (2053), SetInt64Prop (2053)

78.7.45 SetObjectProp

Synopsis: Set the value of an object-type property.

Declaration:
```
procedure SetObjectProp(Instance: TObject; const PropName: string;
                        Value: TObject)
procedure SetObjectProp(Instance: TObject; PropInfo: PPropInfo;
                        Value: TObject)
```

Visibility: default

Description: `SetObjectProp` assigns `Value` to the object property described by `PropInfo` or with name `Propname` for the object `Instance`.

For an example, see GetObjectProp (2038).

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid object property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetObjectProp (2038), SetOrdProp (2055), SetStrProp (2057), SetFloatProp (2053), SetInt64Prop (2053), SetMethodProp (2054)

78.7.46 SetOrdProp

Synopsis: Set value of an ordinal property.

Declaration:
```
procedure SetOrdProp(Instance: TObject; PropInfo: PPropInfo;
                     Value: Int64)
procedure SetOrdProp(Instance: TObject; const PropName: string;
                     Value: Int64)
```

Visibility: default

Description: `SetOrdProp` assigns `Value` to the ordinal property described by `PropInfo` or with name `Propname` for the object `Instance`.

Ordinal properties that can be set include:

Integers and subranges of integers The actual value of the integer must be passed.

Enumerated types and subranges of enumerated types The ordinal value of the enumerated type must be passed.

Subrange typesof integers or enumerated types. Here the ordinal value must be passed.

SetsIf the base type of the set has less than 31 possible values. For each possible value; the corresponding bit of `Value` must be set.

For an example, see GetOrdProp (2040).

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid ordinal property of `Instance`. No range checking is performed. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetOrdProp (2040), SetStrProp (2057), SetFloatProp (2053), SetInt64Prop (2053), SetMethodProp (2054)

78.7.47 SetPropValue

Synopsis: Set property value as variant.

Declaration:
```
procedure SetPropValue(Instance: TObject; const PropName: string;
                       const Value: Variant)
procedure SetPropValue(Instance: TObject; PropInfo: PPropInfo;
                       const Value: Variant)
```

Visibility: default

Description: Due to missing Variant support, this function is not yet implemented; it is provided for Delphi compatibility only.

78.7.48 SetRawByteStrProp

Synopsis: Set the value of a string property using a `RawByteString`.

Declaration:
```
procedure SetRawByteStrProp(Instance: TObject; const PropName: string;
                            const Value: RawByteString)
procedure SetRawByteStrProp(Instance: TObject; PropInfo: PPropInfo;
                            const Value: RawByteString)
```

Visibility: default

Description: `GetRawbyteStrProp` writes the value of a string property of the object `Instance` to `Value`, a `RawByteString`. This means no codepage translation is done. The property to write can be specified using the name `PropName` or the property info `PropInfo`.

See also: SetStrProp (2057), GetRawByteStrProp (2044)

78.7.49 SetRawInterfaceProp

Synopsis: Set a raw (CORBA) interface property.

Declaration:
```
procedure SetRawInterfaceProp(Instance: TObject;
                              const PropName: string;
                              const Value: Pointer)
procedure SetRawInterfaceProp(Instance: TObject; PropInfo: PPropInfo;
                              const Value: Pointer)
```

Visibility: default

Description: `SetRawInterfaceProp` can be used to set the value of a published CORBA interface with name `PropName` from object `Instance` to `Value`. Alternatively, the required property information can be specified by `PropInfo` instead of the property name. In difference with the SetInterfaceProp (2054) procedure, no reference counting is done.

Errors: If the property `PropName` does not exist, an `EPropertyError` exception is raised.

See also: SetInterfaceProp (2054), GetRawInterfaceProp (2044)

78.7.50 SetSetProp

Synopsis: Set value of set-typed property.

Declaration:
```
procedure SetSetProp(Instance: TObject; const PropName: string;
                     const Value: string)
procedure SetSetProp(Instance: TObject; const PropInfo: PPropInfo;
                     const Value: string)
```

Visibility: default

Description: `SetSetProp` sets the property specified by `PropInfo` or `PropName` for object `Instance` to `Value`. `Value` is a string which contains a comma-separated list of values, each value being a string-representation of the enumerated value that should be included in the set. The value should be accepted by the StringToSet (2059) function.

The value can be formed using the SetToString (2057) function.

For an example, see GetSetProp (2044).

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid ordinal property of `Instance`. No range checking is performed. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetSetProp (2044), SetOrdProp (2055), SetStrProp (2057), SetFloatProp (2053), SetInt64Prop (2053), SetMethodProp (2054), SetToString (2057), StringToSet (2059)

78.7.51 SetStrProp

Synopsis: Set value of a string property.

Declaration:
```
procedure SetStrProp(Instance: TObject; const PropName: string;
                     const Value: AnsiString)
procedure SetStrProp(Instance: TObject; PropInfo: PPropInfo;
                     const Value: Ansistring)
```

Visibility: default

Description: `SetStrProp` assigns `Value` to the string property described by `PropInfo` or with name `Propname` for object `Instance`.

For an example, see GetStrProp (2046)

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid string property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetStrProp (2046), SetWideStrProp (2059), SetOrdProp (2055), SetFloatProp (2053), SetInt64Prop (2053), SetMethodProp (2054)

78.7.52 SetToString

Synopsis: Convert set to a string description.

Declaration:
```
function SetToString(TypeInfo: PTypeInfo; Value: LongInt;
                     Brackets: Boolean) : string
function SetToString(PropInfo: PPropInfo; Value: LongInt;
                     Brackets: Boolean) : string
function SetToString(PropInfo: PPropInfo; Value: LongInt) : string
```

```
function SetToString(TypeInfo: PTypeInfo; Value: Pointer;
                     Brackets: Boolean) : string
function SetToString(PropInfo: PPropInfo; Value: Pointer;
                     Brackets: Boolean) : string
```

Visibility: default

Description: `SetToString` takes an integer representation of a set (as received e.g. by `GetOrdProp`) and turns it into a string representing the elements in the set, based on the type information found in the `PropInfo` property information. By default, the string representation is not surrounded by square brackets. Setting the `Brackets` parameter to `True` will surround the string representation with brackets.

The function returns the string representation of the set.

Errors: No checking is done to see whether `PropInfo` points to valid property information.

See also: GetEnumName (2032), GetEnumValue (2034), StringToSet (2059)

Listing: ./examples/typinfex/ex18.pp

```
program example18;

{ This program demonstrates the SetToString function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;
  I : longint;

begin
  O:=TMyTestObject.Create;
  PI:=GetPropInfo(O,'SetField');
  O.SetField:=[mefirst,meSecond,meThird];
  I:=GetOrdProp(O,PI);
  Writeln('Set property to string : ');
  Writeln('Value  : ',SetToString(PI,I,False));
  O.SetField:=[mefirst,meSecond];
  I:=GetOrdProp(O,PI);
  Writeln('Value  : ',SetToString(PI,I,True));
  I:=StringToSet(PI,'mefirst');
  SetOrdProp(O,PI,I);
  I:=GetOrdProp(O,PI);
  Writeln('Value  : ',SetToString(PI,I,False));
  I:=StringToSet(PI,'[mesecond,methird]');
  SetOrdProp(O,PI,I);
  I:=GetOrdProp(O,PI);
  Writeln('Value  : ',SetToString(PI,I,True));
  O.Free;
end.
```

78.7.53 SetUnicodeStrProp

Synopsis: Set UnicodeString-valued property.

Declaration:
```
procedure SetUnicodeStrProp(Instance: TObject; const PropName: string;
                            const Value: UnicodeString)
procedure SetUnicodeStrProp(Instance: TObject; PropInfo: PPropInfo;
                            const Value: UnicodeString)
```

Visibility: default

Description: `SetUnicodeStrProp` sets the UnicodeString property from `Instance` to `Value`, where the property is identified by the `PropInfo` pointer or the `PropertyName`.

Errors: If no property of the indicated name exists, or it is not of type unicodestring, an exception will occur.

See also: SetStrProp (2057), GetUnicodeStrProp (2047)

78.7.54 SetVariantProp

Synopsis: Set value of a variant property.

Declaration:
```
procedure SetVariantProp(Instance: TObject; const PropName: string;
                         const Value: Variant)
procedure SetVariantProp(Instance: TObject; PropInfo: PPropInfo;
                         const Value: Variant)
```

Visibility: default

Description: Due to missing Variant support, this function is not yet implemented. Provided for Delphi compatibility only.

78.7.55 SetWideStrProp

Synopsis: Set a widestring property.

Declaration:
```
procedure SetWideStrProp(Instance: TObject; const PropName: string;
                         const Value: WideString)
procedure SetWideStrProp(Instance: TObject; PropInfo: PPropInfo;
                         const Value: WideString)
```

Visibility: default

Description: `SetWideStrProp` assigns `Value` to the widestring property described by `PropInfo` or with name `Propname` for object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid widestring property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: GetWideStrProp (2048), SetStrProp (2057), SetOrdProp (2055), SetFloatProp (2053), SetInt64Prop (2053), SetMethodProp (2054)

78.7.56 StringToSet

Synopsis: Convert string description to a set.

Declaration:
```
function StringToSet(PropInfo: PPropInfo; const Value: string) : LongInt
function StringToSet(TypeInfo: PTypeInfo; const Value: string) : LongInt
procedure StringToSet(PropInfo: PPropInfo; const Value: string;
                      Result: Pointer)
```

```
procedure StringToSet(TypeInfo: PTypeInfo; const Value: string;
                      Result: Pointer)
```

Visibility: default

Description: `StringToSet` converts the string representation of a set in `Value` to a integer representation of the set, using the property information found in `PropInfo`. This property information should point to the property information of a set property. The function returns the integer representation of the set. (i.e, the set value, typecast to an integer)

The string representation can be surrounded with square brackets, and must consist of the names of the elements of the base type of the set. The base type of the set should be an enumerated type. The elements should be separated by commas, and may be surrounded by spaces. each of the names will be fed to the GetEnumValue (2034) function.

For an example, see SetToString (2057).

Errors: No checking is done to see whether `PropInfo` points to valid property information. If a wrong name is given for an enumerated value, then an `EPropertyError` will be raised.

See also: GetEnumName (2032), GetEnumValue (2034), SetToString (2057)

78.8 TArrayTypeData

```
TArrayTypeData = packed record
private
  function GetElType : PTypeInfo
  ;
  function GetDims(aIndex: Byte) : PTypeInfo;
public
  property
  ElType : PTypeInfo;
  property Dims[Index: Byte]: PTypeInfo;
  Size
  : SizeInt;
  ElCount : SizeInt;
  ElTypeRef : PPTypeInfo;
  DimCount
  : Byte;
  DimsRef : Array[0..255] of PPTypeInfo;
end
```

`TArrayTypeData` is used to describe arrays in RTTI. It can be encountered when the type kind is `tkArray`, and is used for both static and dynamic arrays and single or multi-dimensional arrays. The type of the array elements is described in `elType`, and the ranges for each of the dimensions (specified in `DimCount` in `Dims`.

78.8.1 Property overview

Page	Properties	Access	Description
2061	Dims	r	Range information for each dimension in the array.
2061	ElType	r	Type information for an element in the array.

78.8.2 TArrayTypeData.ElType

Synopsis: Type information for an element in the array.

Declaration: `Property ElType : PTypeInfo`

Visibility: public

Access: Read

78.8.3 TArrayTypeData.Dims

Synopsis: Range information for each dimension in the array.

Declaration: `Property Dims[Index: Byte]: PTypeInfo`

Visibility: public

Access: Read

78.9 TClassData

```
TClassData = packed record
private
  function GetUnitName : ShortString
  ;
  function GetPropertyTable : PPropData;
public
  ClassType : TClass
  ;
  Parent : PPTypeInfo;
  PropCount : SmallInt;
  property UnitName
  : ShortString;
  property PropertyTable : PPropData;
private
  UnitNameField
  : ShortString;
end
```

`TClassData` describes the memory layout of RTTI data generated for a class.

78.9.1 Property overview

Page	Properties	Access	Description
2062	PropertyTable	r	Property info for this class.
2061	UnitName	r	Name of unit in which class is implemente.

78.9.2 TClassData.UnitName

Synopsis: Name of unit in which class is implemente.

Declaration: `Property UnitName : ShortString`

Visibility: public

Access: Read

Description: `UnitName` returns the name of the unit in which the class is implemented.

78.9.3 TClassData.PropertyTable

Synopsis: Property info for this class.

Declaration: `Property PropertyTable : PPropData`

Visibility: public

Access: Read

Description: `PropertyTable` returns a pointer to a table with property RTTI.

78.10 TInterfaceData

```
TInterfaceData = packed record
private
  function GetUnitName : ShortString
  ;
  function GetPropertyTable : PPropData;
  function GetMethodTable
  : PIntfMethodTable;
public
  Parent : PPTypeInfo;
  Flags : TIntfFlagsBase
  ;
  GUID : TGuid;
  property UnitName : ShortString;
  property PropertyTable
  : PPropData;
  property MethodTable : PIntfMethodTable;
private
  UnitNameField : ShortString;
end
```

`TInterfaceData` describes the memory layout of RTTI data generated for a COM interface. CORBA interfaces are described by TInterfaceRawData (2064).

78.10.1 Property overview

Page	Properties	Access	Description
2063	MethodTable	r	Method info for this interface.
2063	PropertyTable	r	Property info for this interface.
2063	UnitName	r	Unit name.

78.10.2 TInterfaceData.UnitName

Synopsis: Unit name.

Declaration: `Property UnitName : ShortString`

Visibility: public

Access: Read

Description: `UnitName` returns the unit name in which the interface is defined.

78.10.3 TInterfaceData.PropertyTable

Synopsis: Property info for this interface.

Declaration: `Property PropertyTable : PPropData`

Visibility: public

Access: Read

Description: `PropertyTable` returns a pointer to a table with property RTTI.

See also: TInterfaceData.MethodTable (2063)

78.10.4 TInterfaceData.MethodTable

Synopsis: Method info for this interface.

Declaration: `Property MethodTable : PIntfMethodTable`

Visibility: public

Access: Read

Description: `MethodTable` returns a pointer to a table with method RTTI.

See also: TInterfaceData.PropertyTable (2063)

78.11 TInterfaceRawData

```
TInterfaceRawData = packed record
private
  function GetUnitName
  : ShortString;
  function GetIIDStr : ShortString;
  function GetPropertyTable
  : PPropData;
  function GetMethodTable : PIntfMethodTable;
public
  Parent : PPTypeInfo;
  Flags : TIntfFlagsBase;
  IID : TGuid;
  property UnitName : ShortString;
  property IIDStr : ShortString
  ;
```

```
  property PropertyTable : PPropData;
  property MethodTable : PIntfMethodTable
  ;
private
  UnitNameField : ShortString;
end
```

`TRawInterfaceData` describes the memory layout of RTTI data generated for a CORBA interface. COM interfaces are described by TInterfaceData (2062).

78.11.1 Property overview

Page	Properties	Access	Description
2064	IIDStr	r	GUID as string.
2065	MethodTable	r	Method info for this interface.
2064	PropertyTable	r	Property info for this interface.
2064	UnitName	r	Name of unit in which interface is defined.

78.11.2 TInterfaceRawData.UnitName

Synopsis: Name of unit in which interface is defined.

Declaration: `Property UnitName : ShortString`

Visibility: public

Access: Read

Description: `UnitName` returns the unit name in which the interface is defined.

78.11.3 TInterfaceRawData.IIDStr

Synopsis: GUID as string.

Declaration: `Property IIDStr : ShortString`

Visibility: public

Access: Read

Description: `IIDStr` returns a string representation of `IID`.

78.11.4 TInterfaceRawData.PropertyTable

Synopsis: Property info for this interface.

Declaration: `Property PropertyTable : PPropData`

Visibility: public

Access: Read

Description: `PropertyTable` returns a pointer to a table with property RTTI.

See also: TInterfaceRawData.MethodTable (2065)

78.11.5 TInterfaceRawData.MethodTable

Synopsis: Method info for this interface.

Declaration: `Property MethodTable : PIntfMethodTable`

Visibility: public

Access: Read

Description: `MethodTable` returns a pointer to a table with method RTTI.

See also: TInterfaceRawData.PropertyTable (2064)

78.12 TIntfMethodEntry

```
TIntfMethodEntry = packed record
private
  function GetParam(Index
  : Word) : PVmtMethodParam;
  function GetResultLocs : PParameterLocations
  ;
  function GetTail : Pointer;
  function GetNext : PIntfMethodEntry
  ;
  function GetName : ShortString;
public
  ResultType : PPTypeInfo
  ;
  CC : TCallConv;
  Kind : TMethodKind;
  ParamCount : Word;
  StackSize
  : SizeInt;
  NamePtr : PShortString;
  property Name : ShortString
  ;
  property Param[Index: Word]: PVmtMethodParam;
  property ResultLocs
  : PParameterLocations;
  property Tail : Pointer;
  property Next
  : PIntfMethodEntry;
end
```

`TIntfMethodEntry` is a description of the RTTI generated for an `Interface` method. This is different from class or record method RTTI info, which is described by TVmtMethodEntry (2029)

78.12.1 Property overview

Page	Properties	Access	Description
2066	Name	r	Name of the method.
2067	Next	r	Pointer next `TIntfMethodEntry` entry in array.
2066	Param	r	Indexed access to parameter descriptions.
2066	ResultLocs	r	Result location descriptions.
2067	Tail	r	Pointer to location after `TIntfMethodEntry` data.

78.12.2 TIntfMethodEntry.Name

Synopsis: Name of the method.

Declaration: `Property Name : ShortString`

Visibility: public

Access: Read

Description: `Name` dereferences the `NamePtr` field and returns the name of the method.

See also: TIntfMethodEntry.Param (2066)

78.12.3 TIntfMethodEntry.Param

Synopsis: Indexed access to parameter descriptions.

Declaration: `Property Param[Index: Word]: PVmtMethodParam`

Visibility: public

Access: Read

Description: `Param` offers indexed access to the parameter descriptions. It returns the location of the `Index`-th parameter of the method. `Index` must be in the range 0 to `ParamCount-1` (included).

See also: TIntfMethodEntry.ResultLocs (2066)

78.12.4 TIntfMethodEntry.ResultLocs

Synopsis: Result location descriptions.

Declaration: `Property ResultLocs : PParameterLocations`

Visibility: public

Access: Read

Description: `ResultLocs` points to a record with result location descriptors: this describes where the result(s) will be written.

See also: TParameterLocations (2070)

78.12.5 TIntfMethodEntry.Tail

Synopsis: Pointer to location after `TIntfMethodEntry` data.

Declaration: `Property Tail : Pointer`

Visibility: public

Access: Read

Description: `Tail` returns a pointer to the memory location after the `TIntfMethodEntry` data, which can be again a `TIntfMethodEntry` record or other RTTI Data.

See also: TIntfMethodEntry.Next (2067)

78.12.6 TIntfMethodEntry.Next

Synopsis: Pointer next `TIntfMethodEntry` entry in array.

Declaration: `Property Next : PIntfMethodEntry`

Visibility: public

Access: Read

Description: `Next` is the same as TIntfMethodEntry.Tail (2067), but is typed: it returns a pointer to the next `TIntfMethodEntry` record in the RTTI.

See also: TIntfMethodEntry.Tail (2067)

78.13 TIntfMethodTable

```
TIntfMethodTable = packed record
private
  function GetMethod(Index
  : Word) : PIntfMethodEntry;
public
  Count : Word;
  RTTICount : Word
  ;
  property Method[Index: Word]: PIntfMethodEntry;
end
```

`TIntfMethodTable` is a record structure describing all the methods of an interface for which RTTI is generated. It is basically an array of pointers TIntfMethodTable.Method (2068).

78.13.1 Property overview

Page	Properties	Access	Description
2068	Method	r	indexed access to the methods of the interface.

78.13.2 TIntfMethodTable.Method

Synopsis: indexed access to the methods of the interface.

Declaration: `Property Method[Index: Word]: PIntfMethodEntry`

Visibility: public

Access: Read

Description: `Method` offers indexed access to the method descriptions. It returns the location of the `Index`-th method of the interface `Index` must be in the range 0 to `Count-1` (included).

See also: PIntfMethodEntry (2022), TIntfMethodEntry (2065)

78.14 TManagedField

```
TManagedField = packed record
private
  function GetTypeRef : PTypeInfo
  ;
public
  property TypeRef : PTypeInfo;
  TypeRefRef : PPTypeInfo
  ;
  FldOffset : SizeInt;
end
```

`TManagedField` describes 1 managed field in a record. It consists of type information (`TypeRef`) and an offset in the record's memory layout (`FldOffset`). Size can be determined from the type information.

78.14.1 Property overview

Page	Properties	Access	Description
2068	TypeRef	r	Type information for the field.

78.14.2 TManagedField.TypeRef

Synopsis: Type information for the field.

Declaration: `Property TypeRef : PTypeInfo`

Visibility: public

Access: Read

78.15 TParameterLocation

```
TParameterLocation = packed record
private
  LocType : Byte;
  function
```

```
  GetRegType : TRegisterType;
  function GetReference : Boolean;
  function GetShiftVal : Int8;
public
  RegSub : TSubRegister;
  RegNumber
  : Word;
  Offset : SizeInt;
  property Reference : Boolean;
  property
  RegType : TRegisterType;
  property ShiftVal : Int8;
end
```

`TParameterLocation` is used by RTTI to describe the location of a parameter (argument) when arguments are passed to a method call. It has the same memory layout as the actual RTTI data. It offers the following fields and properties:

RegSub Sub register indicator.

RegNumber Register number.

Offset Stack offset or index of register.

Property Reference Is the offset a stack offset or register index.

Property RegType Register type.

Property ShiftVal Register shift.

78.15.1 Property overview

Page	Properties	Access	Description
2069	Reference	r	Is the offset a stack offset or register index.
2070	RegType	r	Register type.
2070	ShiftVal	r	Register shift.

78.15.2 TParameterLocation.Reference

Synopsis: Is the offset a stack offset or register index.

Declaration: `Property Reference : Boolean`

Visibility: public

Access: Read

Description: `Reference` indicates whether the register is the index register (`True`), otherwise the register in wihch (part of) the parameter resides.

See also: TParameterLocation.ShiftVal (2070)

78.15.3 TParameterLocation.RegType

Synopsis: Register type.

Declaration: `Property RegType : TRegisterType`

Visibility: public

Access: Read

Description: `RegType` returns the type of the register location. It is calculated from a private field.

See also: TRegisterType (2027)

78.15.4 TParameterLocation.ShiftVal

Synopsis: Register shift.

Declaration: `Property ShiftVal : Int8`

Visibility: public

Access: Read

Description: `ShiftVal` is zero if Reference (2069) is false, or `Offset` otherwise.

See also: Reference (2069)

78.16 TParameterLocations

```
TParameterLocations = packed record
private
  function GetLocation
  (aIndex: Byte) : PParameterLocation;
  function GetTail : Pointer
  ;
public
  Count : Byte;
  property Location[Index: Byte]: PParameterLocation
  ;
  property Tail : Pointer;
end
```

`TParameterLocations` is used by RTTI to describe the array of locations of method parameters (arguments) when arguments are passed to a method call. It describes the memory layout of the actual RTTI data. It offers the following fields and properties:

78.16.1 Property overview

Page	Properties	Access	Description
2071	Location	r	Indexed access to the various parameter location descriptors.
2071	Tail	r	Memory location after parameter data.

78.16.2 TParameterLocations.Location

Synopsis: Indexed access to the various parameter location descriptors.

Declaration: `Property Location[Index: Byte]: PParameterLocation`

Visibility: public

Access: Read

Description: `Location` offers indexed access to the various TParameterLocation (2069) records that make up a method's parameters description. Note that it returns a pointer to the actual record, not a copy of the record.

See also: PParameterLocations (2022), TParameterLocation (2069), PParameterLocation (2022)

78.16.3 TParameterLocations.Tail

Synopsis: Memory location after parameter data.

Declaration: `Property Tail : Pointer`

Visibility: public

Access: Read

Description: `Tail` points to the memory location after the `TParameterLocations` data (the last TParameterLocation (2069) record). It can be used to get the location of the next structure in the RTTI.

See also: TParameterLocation (2069)

78.17 TProcedureParam

```
TProcedureParam = packed record
private
  function GetParamType :
  PTypeInfo;
  function GetFlags : Byte;
public
  property ParamType
  : PTypeInfo;
  property Flags : Byte;
  ParamFlags : TParamFlags
  ;
  ParamTypeRef : PPTypeInfo;
  Name : ShortString;
end
```

`TProcedureParam` describes a single parameter to a procedure (or function).

78.17.1 Property overview

Page	Properties	Access	Description
2072	Flags	r	Flags for this parameter (see TParamFlags).
2072	ParamType	r	Type information for this parameter.

78.17.2 TProcedureParam.ParamType

Synopsis: Type information for this parameter.

Declaration: `Property ParamType : PTypeInfo`

Visibility: public

Access: Read

78.17.3 TProcedureParam.Flags

Synopsis: Flags for this parameter (see TParamFlags).

Declaration: `Property Flags : Byte`

Visibility: public

Access: Read

78.18 TProcedureSignature

```
TProcedureSignature = packed record
private
  function GetResultType
  : PTypeInfo;
public
  property ResultType : PTypeInfo;
  Flags :
  Byte;
  CC : TCallConv;
  ResultTypeRef : PPTypeInfo;
  ParamCount
  : Byte;
  function GetParam(ParamIndex: Integer) : PProcedureParam
  ;
end
```

`TProcedureSignature` describes a procedure/method call signature. It consists of some flags (`Flags`), a calling convention (`CC`), the result type (`ResultType`) if any, and a list of `ParamCount` parameters (of type TProcedureParam (2071)).

78.18.1 Method overview

Page	Method	Description
2073	GetParam	Get parameter signature.

78.18.2 Property overview

Page	Properties	Access	Description
2073	ResultType	r	Result type info (Nil if no result).

78.18.3 TProcedureSignature.GetParam

Synopsis: Get parameter signature.

Declaration: `function GetParam(ParamIndex: Integer) : PProcedureParam`

Visibility: public

Description: `GetParam` can be used to retrieve a pointer to the description of a parameter. The index `ParamIndex` is zero-based.

Errors: In case of an invalid parameter index, `Nil` is returned.

See also: TProcedureParam (2071)

78.18.4 TProcedureSignature.ResultType

Synopsis: Result type info (Nil if no result).

Declaration: `Property ResultType : PTypeInfo`

Visibility: public

Access: Read

78.19 TPropData

```
TPropData = packed record
private
  function GetProp(Index: Word)
  : PPropInfo;
  function GetTail : Pointer;
public
  PropCount : Word
  ;
  PropList : record
  public
    _alignmentdummy : PtrInt;
  end
  ;
  property Prop[Index: Word]: PPropInfo;
  property Tail : Pointer
  ;
end
```

The `TPropData` record is not used, but is provided for completeness and compatibility with Delphi.

78.19.1 Property overview

Page	Properties	Access	Description
2074	Prop	r	Indexed access to property type info.
2074	Tail	r	Pointer to next RTTI structure.

78.19.2 TPropData.Prop

Synopsis: Indexed access to property type info.

Declaration: `Property Prop[Index: Word]: PPropInfo`

Visibility: public

Access: Read

Description: `Prop` provides indexed access to the property type info of all properties of a class. Valid values for `Index` are in the range 0..`WordCount-1`

See also: PPropInfo (2022), TPropInfo (2075)

78.19.3 TPropData.Tail

Synopsis: Pointer to next RTTI structure.

Declaration: `Property Tail : Pointer`

Visibility: public

Access: Read

Description: `Tail` points to the memory area after the last property info. It can be used to get the next RTTI info.

78.20 TPropInfo

```
TPropInfo = packed record
private
  function GetPropType : PTypeInfo
  ;
  function GetTail : Pointer;
  function GetNext : PPropInfo;
public
  PropTypeRef : PPTypeInfo;
  GetProc : CodePointer;
  SetProc :
  CodePointer;
  StoredProc : CodePointer;
  Index : Integer;
  Default
  : LongInt;
  NameIndex : SmallInt;
  PropProcs : Byte;
  Name : ShortString
  ;
  property PropType : PTypeInfo;
  property Tail : Pointer;
  property
  Next : PPropInfo;
end
```

The TPropInfo record describes one published property of a class. The property information of a class are stored as an array of TPropInfo records.

The Name field is stored not with 255 characters, but with just as many characters as required to store the name.

78.20.1 Property overview

Page	Properties	Access	Description
2075	Next	r	Typed pointer TPropInfo record after current TPropInfo data.
2075	PropType	r	Property type.
2075	Tail	r	Pointer to memory location after TPropInfo data.

78.20.2 TPropInfo.PropType

Synopsis: Property type.

Declaration: `Property PropType : PTypeInfo`

Visibility: public

Access: Read

78.20.3 TPropInfo.Tail

Synopsis: Pointer to memory location after TPropInfo data.

Declaration: `Property Tail : Pointer`

Visibility: public

Access: Read

Description: Tail points to the memory location right after the TPropInfo data. This can be another TPropInfo record, or a completely different kind of information.

See also: TPropInfo.Next (2075)

78.20.4 TPropInfo.Next

Synopsis: Typed pointer TPropInfo record after current TPropInfo data.

Declaration: `Property Next : PPropInfo`

Visibility: public

Access: Read

Description: Next points to the same memory location as TPropInfo.Tail (2075), but is typed. It should not be used on the last element of a TPropInfo array.

See also: TPropInfo.Tail (2075)

78.21 TTypeData

```
TTypeData = packed record
private
  function GetBaseType : PTypeInfo
  ;
  function GetCompType : PTypeInfo;
  function GetParentInfo : PTypeInfo
  ;
  function GetRecInitData : PRecInitData;
  function GetHelperParent
  : PTypeInfo;
  function GetExtendedInfo : PTypeInfo;
  function GetIntfParent
  : PTypeInfo;
  function GetRawIntfParent : PTypeInfo;
  function
  GetIIDStr : ShortString;
  function GetElType : PTypeInfo;
  function
  GetElType2 : PTypeInfo;
  function GetInstanceType : PTypeInfo;
  function GetRefType : PTypeInfo;
public
  property BaseType : PTypeInfo
  ;
  property CompType : PTypeInfo;
  property ParentInfo : PTypeInfo
  ;
  property RecInitData : PRecInitData;
  property HelperParent
  : PTypeInfo;
  property ExtendedInfo : PTypeInfo;
  property IntfParent
  : PTypeInfo;
  property RawIntfParent : PTypeInfo;
  property IIDStr
  : ShortString;
  property ElType2 : PTypeInfo;
  property ElType
  : PTypeInfo;
  property InstanceType : PTypeInfo;
  property RefType
  : PTypeInfo;
case TTypeKind of
tkUnKnown: (
);, tkLString: (
);,
  tkWString: (
);, tkVariant: (
);, tkUString: (
);
tkAString: (
```

```
public
  CodePage : Word;
);
tkInt64: (
public
  OrdType : TOrdType;
case
  TTypeKind of
tkInteger: (
public
  MinValue : LongInt;
  MaxValue
  : LongInt;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef
  : PPTypeInfo;
  NameList : ShortString;
);
);, tkChar: (
public
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration
  : (
public
  BaseTypeRef : PPTypeInfo;
  NameList : ShortString;
  );
);, tkEnumeration: (
public
  MinValue : LongInt;
  MaxValue :
  LongInt;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef
  : PPTypeInfo;
  NameList : ShortString;
);
);, tkBool: (
public
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration
  : (
public
  BaseTypeRef : PPTypeInfo;
  NameList : ShortString;
  );
```

```
);, tkWChar: (
public
  MinValue : LongInt;
  MaxValue : LongInt
  ;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef : PPTypeInfo
  ;
  NameList : ShortString;
);
);
tkInt64: (
public
  MinInt64Value
  : Int64;
  MaxInt64Value : Int64;
);
tkQWord: (
public
  MinQWordValue
  : QWord;
  MaxQWordValue : QWord;
);
tkSet: (
public
  SetSize :
  SizeInt;
  CompTypeRef : PPTypeInfo;
);
);, tkQWord: (
public
  OrdType
  : TOrdType;
case TTypeKind of
tkInteger: (
public
  MinValue : LongInt
  ;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef : PPTypeInfo;
  NameList : ShortString;
);
);, tkChar
  : (
public
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind
  of
```

```
tkEnumeration: (
public
  BaseTypeRef : PPTypeInfo;
  NameList
  : ShortString;
);
);, tkEnumeration: (
public
  MinValue : LongInt
  ;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef : PPTypeInfo;
  NameList : ShortString;
);
);, tkBool
  : (
public
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind
  of
tkEnumeration: (
public
  BaseTypeRef : PPTypeInfo;
  NameList
  : ShortString;
);
);, tkWChar: (
public
  MinValue : LongInt;
  MaxValue
  : LongInt;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef
  : PPTypeInfo;
  NameList : ShortString;
);
);
tkInt64: (
public
  MinInt64Value : Int64;
  MaxInt64Value : Int64;
);
tkQWord: (
public
  MinQWordValue : QWord;
  MaxQWordValue : QWord;
);
tkSet: (
```

```
public
  SetSize : SizeInt;
  CompTypeRef : PPTypeInfo;
);
);, tkInteger
  : (
public
  OrdType : TOrdType;
case TTypeKind of
tkInteger: (
public
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration
  : (
public
  BaseTypeRef : PPTypeInfo;
  NameList : ShortString;
  );
);, tkChar: (
public
  MinValue : LongInt;
  MaxValue : LongInt
  ;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef : PPTypeInfo
  ;
  NameList : ShortString;
);
);, tkEnumeration: (
public
  MinValue
  : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration
  : (
public
  BaseTypeRef : PPTypeInfo;
  NameList : ShortString;
  );
);, tkBool: (
public
  MinValue : LongInt;
  MaxValue : LongInt
  ;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef : PPTypeInfo
  ;
```

```
  NameList : ShortString;
);
);, tkWChar: (
public
  MinValue :
  LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration:
  (
public
  BaseTypeRef : PPTypeInfo;
  NameList : ShortString;
);
  );
tkInt64: (
public
  MinInt64Value : Int64;
  MaxInt64Value : Int64
  ;
);
tkQWord: (
public
  MinQWordValue : QWord;
  MaxQWordValue :
  QWord;
);
tkSet: (
public
  SetSize : SizeInt;
  CompTypeRef : PPTypeInfo
  ;
);
);, tkChar: (
public
  OrdType : TOrdType;
case TTypeKind of
  tkInteger: (
public
  MinValue : LongInt;
  MaxValue : LongInt;
case
  TTypeKind of
tkEnumeration: (
public
  BaseTypeRef : PPTypeInfo;
  NameList : ShortString;
);
);, tkChar: (
public
  MinValue : LongInt
  ;
  MaxValue : LongInt;
case TTypeKind of
```

```
tkEnumeration: (
public
  BaseTypeRef : PPTypeInfo;
  NameList : ShortString;
);
);, tkEnumeration
  : (
public
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind
  of
tkEnumeration: (
public
  BaseTypeRef : PPTypeInfo;
  NameList
  : ShortString;
);
);, tkBool: (
public
  MinValue : LongInt;
  MaxValue
  : LongInt;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef
  : PPTypeInfo;
  NameList : ShortString;
);
);, tkWChar: (
public
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration
  : (
public
  BaseTypeRef : PPTypeInfo;
  NameList : ShortString;
  );
);
tkInt64: (
public
  MinInt64Value : Int64;
  MaxInt64Value
  : Int64;
);
tkQWord: (
public
  MinQWordValue : QWord;
  MaxQWordValue
  : QWord;
);
```

```
tkSet: (
public
  SetSize : SizeInt;
  CompTypeRef :
  PPTypeInfo;
);
);, tkEnumeration: (
public
  OrdType : TOrdType;
  case TTypeKind of
tkInteger: (
public
  MinValue : LongInt;
  MaxValue
  : LongInt;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef
  : PPTypeInfo;
  NameList : ShortString;
);
);, tkChar: (
public
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration
  : (
public
  BaseTypeRef : PPTypeInfo;
  NameList : ShortString;
  );
);, tkEnumeration: (
public
  MinValue : LongInt;
  MaxValue :
  LongInt;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef
  : PPTypeInfo;
  NameList : ShortString;
);
);, tkBool: (
public
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration
  : (
public
  BaseTypeRef : PPTypeInfo;
```

```
  NameList : ShortString;
  );
);, tkWChar: (
public
  MinValue : LongInt;
  MaxValue : LongInt
  ;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef : PPTypeInfo
  ;
  NameList : ShortString;
);
);
tkInt64: (
public
  MinInt64Value
  : Int64;
  MaxInt64Value : Int64;
);
tkQWord: (
public
  MinQWordValue
  : QWord;
  MaxQWordValue : QWord;
);
tkSet: (
public
  SetSize :
  SizeInt;
  CompTypeRef : PPTypeInfo;
);
);, tkBool: (
public
  OrdType
  : TOrdType;
case TTypeKind of
tkInteger: (
public
  MinValue : LongInt
  ;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef : PPTypeInfo;
  NameList : ShortString;
);
);, tkChar
  : (
public
  MinValue : LongInt;
  MaxValue : LongInt;
```

```
case TTypeKind
  of
tkEnumeration: (
public
  BaseTypeRef : PPTypeInfo;
  NameList
  : ShortString;
);
);, tkEnumeration: (
public
  MinValue : LongInt
  ;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef : PPTypeInfo;
  NameList : ShortString;
);
);, tkBool
  : (
public
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind
  of
tkEnumeration: (
public
  BaseTypeRef : PPTypeInfo;
  NameList
  : ShortString;
);
);, tkWChar: (
public
  MinValue : LongInt;
  MaxValue
  : LongInt;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef
  : PPTypeInfo;
  NameList : ShortString;
);
);
tkInt64: (
public
  MinInt64Value : Int64;
  MaxInt64Value : Int64;
);
tkQWord: (
public
  MinQWordValue : QWord;
  MaxQWordValue : QWord;
```

```
);
tkSet: (
public
  SetSize : SizeInt;
  CompTypeRef : PPTypeInfo;
);
);, tkWChar:
  (
public
  OrdType : TOrdType;
case TTypeKind of
tkInteger: (
public
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration
  : (
public
  BaseTypeRef : PPTypeInfo;
  NameList : ShortString;
  );
);, tkChar: (
public
  MinValue : LongInt;
  MaxValue : LongInt
  ;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef : PPTypeInfo
  ;
  NameList : ShortString;
);
);, tkEnumeration: (
public
  MinValue
  : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration
  : (
public
  BaseTypeRef : PPTypeInfo;
  NameList : ShortString;
  );
);, tkBool: (
public
  MinValue : LongInt;
  MaxValue : LongInt
  ;
case TTypeKind of
tkEnumeration: (
public
```

```
  BaseTypeRef : PPTypeInfo
  ;
  NameList : ShortString;
);
);, tkWChar: (
public
  MinValue :
  LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration:
  (
public
  BaseTypeRef : PPTypeInfo;
  NameList : ShortString;
);
  );
tkInt64: (
public
  MinInt64Value : Int64;
  MaxInt64Value : Int64
  ;
);
tkQWord: (
public
  MinQWordValue : QWord;
  MaxQWordValue :
  QWord;
);
tkSet: (
public
  SetSize : SizeInt;
  CompTypeRef : PPTypeInfo
  ;
);
);, tkSet: (
public
  OrdType : TOrdType;
case TTypeKind of
tkInteger
  : (
public
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind
  of
tkEnumeration: (
public
  BaseTypeRef : PPTypeInfo;
  NameList
  : ShortString;
);
);, tkChar: (
public
```

```
  MinValue : LongInt;
  MaxValue
  : LongInt;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef
  : PPTypeInfo;
  NameList : ShortString;
);
);, tkEnumeration: (
public
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration
  : (
public
  BaseTypeRef : PPTypeInfo;
  NameList : ShortString;
  );
);, tkBool: (
public
  MinValue : LongInt;
  MaxValue : LongInt
  ;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef : PPTypeInfo
  ;
  NameList : ShortString;
);
);, tkWChar: (
public
  MinValue :
  LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration:
  (
public
  BaseTypeRef : PPTypeInfo;
  NameList : ShortString;
);
  );
tkInt64: (
public
  MinInt64Value : Int64;
  MaxInt64Value : Int64
  ;
);
tkQWord: (
public
```

```
  MinQWordValue : QWord;
  MaxQWordValue :
  QWord;
);
tkSet: (
public
  SetSize : SizeInt;
  CompTypeRef : PPTypeInfo
  ;
);
);
tkFloat: (
public
  FloatType : TFloatType;
);
tkSString:
  (
public
  MaxLength : Byte;
);
tkClass: (
public
  ClassType : TClass
  ;
  ParentInfoRef : PPTypeInfo;
  PropCount : SmallInt;
  UnitName
  : ShortString;
);
tkRecord: (
public
  RecInitInfo : Pointer;
  RecSize
  : Integer;
case Boolean of
False: (
public
  ManagedFldCount : Integerdeprecated
  ;
);
True: (
public
  TotalFieldCount : Integer;
);
);
tkHelper:
  (
public
  HelperParentRef : PPTypeInfo;
  ExtendedInfoRef : PPTypeInfo
  ;
  HelperProps : SmallInt;
  HelperUnit : ShortString;
);
```

```
tkMethod
  : (
public
  MethodKind : TMethodKind;
  ParamCount : Byte;
  ParamList
  : Array[0..1023] of Char;
);
tkProcVar: (
public
  ProcSig : TProcedureSignature
  ;
);
tkInterface: (
public
  IntfParentRef : PPTypeInfo;
  IntfFlags
  : TIntfFlagsBase;
  GUID : TGuid;
  IntfUnit : ShortString;
);
tkInterfaceRaw
  : (
public
  RawIntfParentRef : PPTypeInfo;
  RawIntfFlags : TIntfFlagsBase
  ;
  IID : TGuid;
  RawIntfUnit : ShortString;
);
tkArray: (
public
  ArrayData : TArrayTypeData;
);
tkDynArray: (
public
  elSize :
  PtrUInt;
  elType2Ref : PPTypeInfo;
  varType : LongInt;
  elTypeRef
  : PPTypeInfo;
  DynUnitName : ShortStringBase;
);
tkClassRef: (
public
  InstanceTypeRef : PPTypeInfo;
);
tkPointer: (
public
  RefTypeRef
  : PPTypeInfo;
);
end
```

If the typeinfo kind is `tkClass`, then the property information follows the `UnitName` string, as an array of TPropInfo (2075) records.

78.21.1 Property overview

Page	Properties	Access	Description
2091	BaseType	r	Base type on which this type is based.
2091	CompType	r	Comp type.
2093	ElType	r	Dynamical array Element type.
2093	ElType2	r	Dynamical array Element type.
2092	ExtendedInfo	r	Type information for the extended type (record or class).
2092	HelperParent	r	Type information for parent helper class.
2093	IIDStr	r	IID string representation of interface.
2093	InstanceType	r	Pointer to instance type data.
2092	IntfParent	r	Parent interface type data.
2091	ParentInfo	r	Parent class type info.
2093	RawIntfParent	r	Raw interface parent.
2092	RecInitData	r	Pointer to record initialization data.
2093	RefType	r	Type info for referred type (Nil for untyped pointer).

78.21.2 TTypeData.BaseType

Synopsis: Base type on which this type is based.

Declaration: `Property BaseType : PTypeInfo`

Visibility: public

Access: Read

78.21.3 TTypeData.CompType

Synopsis: Comp type.

Declaration: `Property CompType : PTypeInfo`

Visibility: public

Access: Read

78.21.4 TTypeData.ParentInfo

Synopsis: Parent class type info.

Declaration: `Property ParentInfo : PTypeInfo`

Visibility: public

Access: Read

78.21.5 TTypeData.RecInitData

Synopsis: Pointer to record initialization data.

Declaration: `Property RecInitData : PRecInitData`

Visibility: public

Access: Read

Description: `RecInitData` is a pointer to a TRecInitData (2027) record. It is only valid if the type data describes a record.

See also: TRecInitData (2027)

78.21.6 TTypeData.HelperParent

Synopsis: Type information for parent helper class.

Declaration: `Property HelperParent : PTypeInfo`

Visibility: public

Access: Read

Description: `HelperParent` points to the type information of the parent helper class. It is `Nil` if there is no parent class.

See also: TTypeData.ExtendedInfo (2092)

78.21.7 TTypeData.ExtendedInfo

Synopsis: Type information for the extended type (record or class).

Declaration: `Property ExtendedInfo : PTypeInfo`

Visibility: public

Access: Read

Description: `ExtendedInfo` points to the type information of the type that is being extended.

See also: TTypeData.HelperParent (2092)

78.21.8 TTypeData.IntfParent

Synopsis: Parent interface type data.

Declaration: `Property IntfParent : PTypeInfo`

Visibility: public

Access: Read

78.21.9 TTypeData.RawIntfParent

Synopsis: Raw interface parent.

Declaration: `Property RawIntfParent : PTypeInfo`

Visibility: public

Access: Read

78.21.10 TTypeData.IIDStr

Synopsis: IID string representation of interface.

Declaration: `Property IIDStr : ShortString`

Visibility: public

Access: Read

78.21.11 TTypeData.ElType2

Synopsis: Dynamical array Element type.

Declaration: `Property ElType2 : PTypeInfo`

Visibility: public

Access: Read

78.21.12 TTypeData.ElType

Synopsis: Dynamical array Element type.

Declaration: `Property ElType : PTypeInfo`

Visibility: public

Access: Read

78.21.13 TTypeData.InstanceType

Synopsis: Pointer to instance type data.

Declaration: `Property InstanceType : PTypeInfo`

Visibility: public

Access: Read

78.21.14 TTypeData.RefType

Synopsis: Type info for referred type (Nil for untyped pointer).

Declaration: `Property RefType : PTypeInfo`

Visibility: public

Access: Read

78.22 TVmtFieldEntry

```
TVmtFieldEntry = packed record
private
  function GetNext : PVmtFieldEntry
  ;
  function GetTail : Pointer;
public
  FieldOffset : PtrUInt;
  TypeIndex : Word;
  Name : ShortString;
  property Tail : Pointer
  ;
  property Next : PVmtFieldEntry;
end
```

`TVmtFieldEntry` records are generated by the compiler for all fields of a record or class that have RTTI associated with them. They describe the field as known to the compiler.

78.22.1 Property overview

Page	Properties	Access	Description
2094	Next	r	Pointer to the next `TVmtFieldEntry`.
2094	Tail	r	Pointer to the next RTTI structure.

78.22.2 TVmtFieldEntry.Tail

Synopsis: Pointer to the next RTTI structure.

Declaration: `Property Tail : Pointer`

Visibility: public

Access: Read

Description: `Tail` points to the memory location after the `TVmtFieldEntry` record. This can be another `TVmtFieldEntry` record or an entirely different data structure.

See also: TVmtFieldEntry.Next (2094)

78.22.3 TVmtFieldEntry.Next

Synopsis: Pointer to the next `TVmtFieldEntry`.

Declaration: `Property Next : PVmtFieldEntry`

Visibility: public

Access: Read

Description: `Next` is the same as TVmtFieldEntry.Tail (2094), but is typed. Note that on the last `TVmtFieldEntry` in the array, the next element is not a `TVmtFieldEntry`.

See also: TVmtFieldEntry.Tail (2094)

78.23 TVmtFieldTable

```
TVmtFieldTable = packed record
private
  function GetField(aIndex
  : Word) : PVmtFieldEntry;
public
  Count : Word;
  ClassTab : PVmtFieldClassTab
  ;
  Fields : Array[0..0] of TVmtFieldEntry;
  property Field[aIndex
  : Word]: PVmtFieldEntry;
end
```

`TVmtFieldTable` describes the fields for which RTTI was generated. A `TVmtFieldTable` entry is generated by the compiler in the RTTI information, it is not something one creates manually. Basically it contains a list of TVmtFieldEntry (2094) values.

78.23.1 Property overview

Page	Properties	Access	Description
2095	Field	r	Indexed access to the fields in the table.

78.23.2 TVmtFieldTable.Field

Synopsis: Indexed access to the fields in the table.

Declaration: `Property Field[aIndex: Word]: PVmtFieldEntry`

Visibility: public

Access: Read

Description: `Field` returns the location of the `aIndex`the field in the table.

See also: TVmtFieldEntry (2094)

78.24 TVmtMethodParam

```
TVmtMethodParam = packed record
private
  function GetTail : Pointer
  ;
  function GetNext : PVmtMethodParam;
  function GetName : ShortString
  ;
public
  ParamType : PPTypeInfo;
  Flags : TParamFlags;
  NamePtr
  : PShortString;
  ParaLocs : PParameterLocations;
```

```
  property Name
  : ShortString;
  property Tail : Pointer;
  property Next : PVmtMethodParam
  ;
end
```

`TVmtMethodParam` describes the type and location of a parameter of a method.

78.24.1 Property overview

Page	Properties	Access	Description
2096	Name	r	Parameter name.
2096	Next	r	Pointer to the next `TVmtMethodParam` record.
2096	Tail	r	Pointer to location after `TVmtMethodParam` data.

78.24.2 TVmtMethodParam.Name

Synopsis: Parameter name.

Declaration: `Property Name : ShortString`

Visibility: public

Access: Read

Description: `Name` is an ease-of-access property for the `NamePtr` field, it simply returns the name as a string.

78.24.3 TVmtMethodParam.Tail

Synopsis: Pointer to location after `TVmtMethodParam` data.

Declaration: `Property Tail : Pointer`

Visibility: public

Access: Read

Description: `Tail` points to the memory location just after the `TVmtMethodParam` structure. It can be used to get access to the next RTTI data, which can be again a `TVmtMethodParam` structure or another structure.

See also: TVmtMethodParam.Next (2096)

78.24.4 TVmtMethodParam.Next

Synopsis: Pointer to the next `TVmtMethodParam` record.

Declaration: `Property Next : PVmtMethodParam`

Visibility: public

Access: Read

Description: `Next` is the same as TVmtMethodParam.Tail (2096), but is typed: it returns a pointer to the next `TVmtMethodParam` record in the RTTI.

See also: TVmtMethodParam.Tail (2096)

78.25 TVmtMethodTable

```
TVmtMethodTable = packed record
private
  function GetEntry(Index
  : LongWord) : PVmtMethodEntry;
public
  Count : LongWord;
  property
  Entry[Index: LongWord]: PVmtMethodEntry;
private
  Entries : Array
  [0..0] of TVmtMethodEntry;
end
```

`TVmtMethodTable` is a record structure describing all the VMT methods of a class. It is basically an array of pointers TVmtMethodTable.Entry (2097).

78.25.1 Property overview

Page	Properties	Access	Description
2097	Entry	r	Indexed access to VMT method information.

78.25.2 TVmtMethodTable.Entry

Synopsis: Indexed access to VMT method information.

Declaration: `Property Entry[Index: LongWord]: PVmtMethodEntry`

Visibility: public

Access: Read

Description: `Method` offers indexed access to the method descriptions. It returns the location of the `Index`-th method of the VMT table `Index` must be in the range 0 to `Count-1` (included).

See also: PVmtMethodEntry (2023), TVmtMethodEntry (2029)

78.26 EPropertyConvertError

78.26.1 Description

`EPropertyConvertError` is not used in the Free Pascal implementation of the typinfo unit, but is declared for Delphi compatibility.

78.27 EPropertyError

78.27.1 Description

Exception raised in case of an error in one of the functions.

Chapter 79

Reference for unit 'unicodedata'

79.1 Used units

Table 79.1: Used units by unit 'unicodedata'

Name	Page
System	1362

79.2 Overview

The fpwidestring (2098) unit relies on having relevant Unicode collation data linked in the binary. The Unicode data is managed using the routines in the unicodedata unit. The FPC project distributes some Unicode collation data in `.bco` files which can be loaded using the LoadCollation (2109) routines. The `LoadCollation` is the main routine of this unit.

All collation data requires at least the Default Unicode Collation Element Table to be registered (called `DUCET`). The DUCET encoding is provided by the unicodeducet unit, part of the rtl-Unicode package.

There are two ways to register collations :

1. **at compile time**: by including the desired collation unit, for example for Russian and Japanese languages to be available you will have to include collation_ru and collation_ja from package "rtl-unicode".
2. **at runtime** using the above mentioned `LoadCollation` function.

The two ways can co-exist: some collations may be compile time included (for example for most used collations) and others can be loaded at runtime in the same application.

The binary collation files are endian sensitive:

- there are files for little endian systems named collation__le.bco (such as collation_ru_le.bco and collation_ja_le.bco)
- there are files for big endian systems named collation__be.bco (such as collation_ru_be.bco and collation_ja_be.bco).

Note that the compile time units collation units (collation_**lang**.pas) include already the unicodeducet.pas (DUCET) unit so it is not necessary to include it manually, contrary to the binary files. So an application that only uses the binary collation files should at least include the unicodeducet unit or manually load the binary collation collation_ducet_le.bco or collation_ducet_be.bco, depending on the endianness of the platform. The LoadCollation (2109) call using a directory and the language `ducet` automatically select the correct file.

79.3 Constants, types and variables

79.3.1 Resource strings

```
SCollationNotFound = 'Collation not found : "%s".'
```

Error message when a collation is not found.

79.3.2 Constants

```
DEFAULT_UCA_COMPARISON_STRENGTH = 3
```

```
DirectorySeparator = '/'
```

```
ENDIAN_SUFFIX : Array[TEndianKind] of UnicodeString = ('le', 'be'
  )
```

`ENDIAN_SUFFIX` contains the suffixes used in LoadCollation (2109) when constructing a collation filename for a language.

```
ERROR_INVALID_CODEPOINT_SEQUENCE = 1
```

Error exit code for UnicodeToLower/UnicodeToUpper.

```
HIGH_SURROGATE_BEGIN = Word($D800)
```

First value for high surrogate values.

```
HIGH_SURROGATE_COUNT = HIGH_SURROGATE_END - HIGH_SURROGATE_BEGIN
  + 1
```

Number of high surrogate values.

```
HIGH_SURROGATE_END = Word($DBFF)
```

Last value for high surrogate values.

```
LOW_SURROGATE_BEGIN = Word($DC00)
```

First value for low surrogate values.

```
LOW_SURROGATE_END = Word($DFFF)
```

Last value for low surrogate values.

```
MAX_LEGAL_UTF32 = $10FFFF
```

Maximum value of a legally encoded UTF32 value (currently unused).

```
MAX_WORD = High(Word)
```

MAX_WORD is the maximum value of a WORD typed value.

```
reCodesetConversion = reRangeError
```

```
ROOT_COLLATION_NAME = 'DUCET'
```

Name of the root collation (used if no collation name is given).

```
UCS4_HALF_BASE = LongWord($10000)
```

Offset for UCS4 character encoding.

```
UCS4_HALF_MASK = Word($3FF)
```

Unused currently.

```
UGC_ClosePunctuation = 14
```

Unicode token category: Close punctuation.

```
UGC_CombiningMark = 6
```

Unicode token category: Uppercase letter.

```
UGC_ConnectPunctuation = 11
```

Unicode token category: Connect punctuation.

```
UGC_Control = 25
```

Unicode token category: control token.

```
UGC_CurrencySymbol = 19
```

Unicode token category: Currency symbol.

```
UGC_DashPunctuation = 12
```

Unicode token category: Dash punctuation.

```
UGC_DecimalNumber = 8
```

Unicode token category: Uppercase letter.

```
UGC_EnclosingMark = 7
```

Unicode token category: Uppercase letter.

```
UGC_FinalPunctuation = 16
```

Unicode token category: Final punctuation.

```
UGC_Format = 26
```

Unicode token category: Formatting token.

```
UGC_InitialPunctuation = 15
```

Unicode token category: Initial punctuation.

```
UGC_LetterNumber = 9
```

Unicode token category: Letter-number.

```
UGC_LineSeparator = 23
```

Unicode token category: Line separator.

```
UGC_LowercaseLetter = 1
```

Unicode general category: Lowercase letter.

```
UGC_MathSymbol = 18
```

Unicode token category: Math symbol.

```
UGC_ModifierLetter = 3
```

Unicode general category: modifier letter.

```
UGC_ModifierSymbol = 20
```

Unicode token category: modifier symbol.

```
UGC_NonSpacingMark = 5
```

Unicode general category: Non-spacing mark.

```
UGC_OpenPunctuation = 13
```

Unicode token category: Open punctuation.

```
UGC_OtherLetter = 4
```

Unicode general category: Other letter.

```
UGC_OtherNumber = 10
```

Unicode token category: Other number.

```
UGC_OtherPunctuation = 17
```

Unicode token category: Other punctuation.

```
UGC_OtherSymbol = 21
```

Unicode token category: Other symbol.

```
UGC_ParagraphSeparator = 24
```

Unicode token category: Paragraph separator.

```
UGC_PrivateUse = 28
```

Unicode token category: For private use.

```
UGC_SpaceSeparator = 22
```

Unicode token category: Space separator.

```
UGC_Surrogate = 27
```

Unicode token category: Surrogate token.

```
UGC_TitlecaseLetter = 2
```

Unicode general category: Titlecase letter.

```
UGC_Unassigned = 29
```

Unicode token category: As yet unassigned.

```
UGC_UppercaseLetter = 0
```

Unicode general category: Uppercase letter.

```
UnicodeCategoryNames : Array[0..29] of string = ('Lu', 'Ll', 'Lt'
  , 'Lm', 'Lo', 'Mn', 'Mc', 'Me', 'Nd', 'Nl', 'No', 'Pc', 'Pd', 'Ps'
  , 'Pe', 'Pi', 'Pf', 'Po', 'Sm', 'Sc', 'Sk', 'So', 'Zs', 'Zl', 'Zp'
  , 'Cc', 'Cf', 'Cs', 'Co', 'Cn')


ZERO_UINT24 : UInt24 = (byte2: 0; byte1: 0; byte0: 0)
```

`TUInt24Rec` value with all bytes zero.

79.3.3 Types

```
PCollationTableItem = ^TCollationTableItem
```

```
PUCA_DataBook = ^TUCA_DataBook
```

Pointer to `TUCA_DataBook` type.

```
PUCA_PropItemContextRec = ^TUCA_PropItemContextRec
```

Pointer to `TUCA_PropItemContextRec`.

```
PUCA_PropItemContextTreeNodeRec =
  ^TUCA_PropItemContextTreeNodeRec
```

Pointer to `TUCA_PropItemContextTreeNodeRec`.

```
PUCA_PropItemContextTreeRec = ^TUCA_PropItemContextTreeRec
```

Pointer to `TUCA_PropItemContextTreeRec`.

```
PUCA_PropItemRec = ^TUCA_PropItemRec
```

Pointer to `TUCA_PropItemRec`.

```
PUCA_PropWeights = ^TUCA_PropWeights
```

Pointer to `TUCA_PropWeights`.

```
PUC_Prop = ^TUC_Prop
```

Pointer to `TUC_Prop` record.

```
PUInt24 = ^UInt24
```

Pointer to `TUInt24Rec`.

```
TCategoryMask = Set of
```

```
TCollationField = (BackWard,VariableLowLimit,VariableHighLimit,
  Alternate,Normalization,Strength)
```

Table 79.2: Enumeration values for type TCollationField

Value	Explanation
Alternate	
BackWard	Backwards encoded.
Normalization	
Strength	
VariableHighLimit	Has upper bound on variable weights.
VariableLowLimit	Has lower bound on variable weights.

`TCollationField` describes some properties of the collation data items.

```
TCollationFields = Set of TCollationField
```

Set of TCollationField.

```
TCollationName = Array[0..(128-1)] of Byte
```

Collation name string type (fixed length).

```
TCollationTableItem = record
public
  Collation : PUCA_DataBook;
  Aliases : TUnicodeStringArray;
end
```

```
TCollationTableItemArray = Array of TCollationTableItem
```

```
TCollationVersion = TCollationName
```

```
TEndianKind = (Little,Big)
```

Table 79.3: Enumeration values for type TEndianKind

Value	Explanation
Big	Big-endian platform.
Little	Little-endian platform.

TEndianKind is an auxiliary enumerated type to enumerate the endianness of platforms.

```
TSetOfByte = Set of Byte
```

```
TUCASortKey = Array of TUCASortKeyItem
```

Array of TUCASortKeyItem.

```
TUCASortKeyItem = Word
```

Alias for WORD.

```
TUCA_PropWeights = packed record
public
  Weights : Array[0..2] of
  Word;
end
```

TUC_PropWeights describes the weights of collation characteristics of a unicode character. It is an internal structure which should not be used directly, the actual structure is subject to change.

```
TUCA_VariableKind = (ucaShifted,ucaNonIgnorable,ucaBlanked,
  ucaShiftedTrimmed,ucaIgnoreSP)
```

Table 79.4: Enumeration values for type TUCA_VariableKind

Value	Explanation
ucaBlanked	Variable collation elements and any subsequent ignorable collation elements are reset so that all weigh
ucaIgnoreSP	Not implemented (variant of Shifted that reduces the set of variable collation elements to include only
ucaNonIgnorable	Variable collation elements are not reset to be quaternary collation elements.
ucaShifted	Variable collation elements are reset to zero at levels one through three.
ucaShiftedTrimmed	This option is the same as Shifted, except that all trailing FFFFs are trimmed from the sort key.

Options for weighting data.

```
TUnicodeStringArray = Array of UnicodeString
```

```
UInt24 = TUInt24Rec
```

Alias for `TUInt24Rec`.

79.4 Procedures and functions

79.4.1 AddAliasCollation

Declaration:
```
function AddAliasCollation(ACollation: PUCA_DataBook;
                           AALias: UnicodeString) : Boolean
```

Visibility: default

79.4.2 BytesToName

Declaration:
```
function BytesToName(const ABytes: Array of Byte) : UnicodeString
```

Visibility: default

79.4.3 BytesToString

Declaration:
```
function BytesToString(const ABytes: Array of Byte;
                       const AValideChars: TSetOfByte) : UnicodeString
```

Visibility: default

79.4.4 CanonicalOrder

Synopsis: Put unicode string in canonical order.

Declaration:
```
procedure CanonicalOrder(var AString: UnicodeString);  Overload
procedure CanonicalOrder(AStr: PUnicodeChar; const ALength: SizeInt)
                        ;  Overload
```

Visibility: default

Description: `CanonicalOrder` transforms a unicode string `AString` (or the alternate form using a null-terminated `AStr` with length `Alength`) so it is in canonical order (as defined by the unicode specification). A string needs to be in canonical order to be able to compare strings. This function is called as part of NormalizeNFD (2110).

See also: NormalizeNFD (2110)

79.4.5 CompareSortKey

Synopsis: Compare two sort keys.

Declaration:
```
function CompareSortKey(const A: TUCASortKey; const B: TUCASortKey)
                        : Integer;  Overload
function CompareSortKey(const A: TUCASortKey; const B: Array of Word)
                        : Integer;  Overload
```

Visibility: default

Description: `CompareSortKey` compares 2 sort keys `A` and `B`. It returns

- a negative number if `A` comes alphabetically ordered before `B`.
- Zero if `A` is alphabetically identical to `B`.
- A positive number if `A` is alphabetically ordered after `B`.

Sort keys can be constructed from unicode strings using ComputeSortKey (2106).

See also: ComputeSortKey (2106)

79.4.6 ComputeSortKey

Synopsis: Compute the sort key for a string.

Declaration:
```
function ComputeSortKey(const AString: UnicodeString;
                        const ACollation: PUCA_DataBook) : TUCASortKey
                        ;  Overload
function ComputeSortKey(const AStr: PUnicodeChar;
                        const ALength: SizeInt;
                        const ACollation: PUCA_DataBook) : TUCASortKey
                        ;  Overload
```

Visibility: default

Description: `ComputeSortKey` computes the sort key for a unicode string `AString` (or the alternate form using a null-terminated `AStr` with length `Alength`) using the Unicode Collation Algorithm data in `ACollation`. This key can then be used in CompareSortKey (2106) to compare unicode strings.

See also: CompareSortKey (2106)

79.4.7 FilterString

Declaration:
```
function FilterString(const AStr: PUnicodeChar; const ALength: SizeInt;
                      const AExcludedMask: TCategoryMask) : UnicodeString
                      ;  Overload
function FilterString(const AStr: UnicodeString;
                      const AExcludedMask: TCategoryMask) : UnicodeString
                      ;  Overload
```

Visibility: default

79.4.8 FindCollation

Synopsis: Find a collation by name.

Declaration:
```
function FindCollation(AName: UnicodeString) : PUCA_DataBook;  Overload
function FindCollation(const AIndex: Integer) : PUCA_DataBook;  Overload
```

Visibility: default

Description: `FindCollection` searches a collation with name `AName` or index `AIndex` in the list of known collations and returns a pointer to the collation data. If the requested collation is not known, i.e. the name is not found, or the index is out of range, then `Nil` is returned. The valid index range is 0 to `GetCollationCount-1`.

See also: GetCollationCount (2108)

79.4.9 FreeCollation

Synopsis: Free collation data.

Declaration:
```
procedure FreeCollation(AItem: PUCA_DataBook)
```

Visibility: default

Description: `FreeCollation` removes all structures in the collation data from memory. It will not do anything when the header field `Dynamic` is `false`. (collations loaded and registered using LoadCollation (2109) are always dynamic and must be freed).

See also: LoadCollation (2109)

79.4.10 FromUCS4

Synopsis: Convert UCS4 to UNicodeChar.

Declaration:
```
procedure FromUCS4(const AValue: UCS4Char; out AHighS: UnicodeChar;
                   out ALowS: UnicodeChar)
```

Visibility: default

Description: `FromUCS4` converts the UCS4 encoded unicode character `AValue` to a set of unicode (surrogate pair) characters encoded in UTF16: `AHighS`, `ALowS`.

See also: ToUCS4 (2111), UnicodeIsHighSurrogate (2111), UnicodeIsLowSurrogate (2111), UnicodeIsSurrogatePair (2112)

79.4.11 GetCollationCount

Synopsis: Return the number of known collations.

Declaration:
```
function GetCollationCount : Integer
```

Visibility: default

Description: `GetCollationCount` returns the number of registered collations. The collation data can be retrieved by index using the FindCollation (2107). The maximum index is `GetCollationCount-1`.

See also: FindCollation (2107)

79.4.12 GetProps

Synopsis: Get unicode character data.

Declaration:
```
function GetProps(const ACodePoint: Word) : PUC_Prop;  Overload
function GetProps(const AHighS: UnicodeChar; const ALowS: UnicodeChar)
                  : PUC_Prop;  Overload
function GetProps(const ACodePoint: Cardinal) : PUC_Prop;  Overload
```

Visibility: default

Description: `GetProps` returns a pointer to a general unicode character property data structure. The character can be specified using a word or cardinal sized codepoint (`ACodePoint`), or using a UTF16 encoded surrogate pair (`AHighS`,`ALowS`).

The returned pointer must not be freed, it points to part of a static structure.

See also: GetPropUCA (2108)

79.4.13 GetPropUCA

Synopsis: Get unicode collation algorithm properties for a unicode character.

Declaration:
```
function GetPropUCA(const AHighS: UnicodeChar;
                    const ALowS: UnicodeChar; const ABook: PUCA_DataBook)
                     : PUCA_PropItemRec;  Overload
function GetPropUCA(const AChar: UnicodeChar;
                    const ABook: PUCA_DataBook) : PUCA_PropItemRec
                    ;  Overload
```

Visibility: default

Description: `GetPropUCA` returns UCA data for the unicode character (`AChar`) or surrogate pair `AHighS`, `ALowS`) in the collation data book `ABook`. If no data is available, `Nil` is returned.

Errors: if an invalid `ABook` is specified, an access violation may occur.

See also: GetProps (2108)

79.4.14 IncrementalCompareString

Synopsis: Compare 2 strings using the specified collation.

Declaration:
```
function IncrementalCompareString(const AStrA: PUnicodeChar;
                                   const ALengthA: SizeInt;
                                   const AStrB: PUnicodeChar;
                                   const ALengthB: SizeInt;
                                   const ACollation: PUCA_DataBook)
                                    : Integer;  Overload
function IncrementalCompareString(const AStrA: UnicodeString;
                                   const AStrB: UnicodeString;
                                   const ACollation: PUCA_DataBook)
                                    : Integer;  Overload
```

Visibility: default

Description: `IncrementalCompareString` creates 2 compare keys from the strings `AStrA` and `AStrB` using collation data in `ACollation`. The keys are computed only to the point where the two strings differ. This means the keys cannot be reused for other comparisons if the strings differ. The two strings can be specified as a unicode string or as a pointer to a null-terminated character array with a length (`ALengthA` and `ALengthB`). It returns then the result of CompareSortKey (2106).

Errors: None.

See also: ComputeSortKey (2106), CompareSortKey (2106)

79.4.15 LoadCollation

Synopsis: Load a binary collation data file from file.

Declaration:
```
function LoadCollation(const AData: Pointer;
                       const ADataLength: Integer;
                       var AAliases: TUnicodeStringArray) : PUCA_DataBook
                       ;  Overload
function LoadCollation(const AData: Pointer; const ADataLength: Integer)
                        : PUCA_DataBook;  Overload
function LoadCollation(const AFileName: UnicodeString;
                       var AAliases: TUnicodeStringArray) : PUCA_DataBook
                       ;  Overload
function LoadCollation(const AFileName: UnicodeString) : PUCA_DataBook
                       ;  Overload
function LoadCollation(const ADirectory: UnicodeString;
                       const ALanguage: UnicodeString;
                       var AAliases: TUnicodeStringArray) : PUCA_DataBook
                       ;  Overload
function LoadCollation(const ADirectory: UnicodeString;
                       const ALanguage: UnicodeString) : PUCA_DataBook
                       ;  Overload
```

Visibility: default

Description: `LoadCollation` loads collation data from file `AFileName`, or from a memory block `AData` with length `ADataLength`. If successful, it returns a pointer which can be used to register the collation using RegisterCollation (2110). if there is a problem with the data, `Nil` is returned.

The filename can also be specified as a `Directory` and language name `ALanguage`. The latter is prepended with `collation_` and appended with the native endianness of the current platform, and has extension `.bco`

Errors: If the file containing data does not exist or has a size which is less than the encoded header size, `Nil` is returned.

See also: RegisterCollation (2110)

79.4.16 NormalizeNFD

Synopsis: Perform unicode normalization D on a string.

Declaration:
```
function NormalizeNFD(const AString: UnicodeString) : UnicodeString
                        ;  Overload
function NormalizeNFD(const AStr: PUnicodeChar; ALength: SizeInt)
                        : UnicodeString;  Overload
```

Visibility: default

Description: `NormalizeNFD` normalizes the string `AString` (or the alternate form using a null-terminated `AStr` with length `Alength`) to Unicode Normalization Form D. The resulting string can be used to determine equivalence of unicode strings.

See also: CanonicalOrder (2105)

79.4.17 PrepareCollation

Synopsis: Prepare a collation for use in the list.

Declaration:
```
procedure PrepareCollation(ACollation: PUCA_DataBook;
                           const ABaseName: UnicodeString;
                           const AChangedFields: TCollationFields)
```

Visibility: default

Description: `PrepareCollation` will link collation definition `ACollation` to the base collection with name `ABaseName` (if empty, it defaults to the root collation). It will also initialize some fields in the definition, copying them from the base collation, but excludes the fields enumerated in `AChangedFields`.

It should normally not be needed to call this function, it is called as part of LoadCollation (2109).

See also: LoadCollation (2109)

79.4.18 RegisterCollation

Synopsis: Register a new collation.

Declaration:
```
function RegisterCollation(const ACollation: PUCA_DataBook) : Boolean
                           ;  Overload
function RegisterCollation(const ACollation: PUCA_DataBook;
                           const AAliasList: Array of UnicodeString)
                            : Boolean;  Overload
function RegisterCollation(ADirectory: UnicodeString;
                           ALanguage: UnicodeString) : Boolean;  Overload
```

Visibility: default

Description: `RegisterCollation` registers a new collation `ACollation` in the list of known collations. The collation data can be specified directly (`ACollation`) or a name of a language (`ALanguage`) in a directory (`ADirectory`). The latter option will load the binary encoded collation from file using LoadCollation (2109).

If the collation is loaded correctly, `True` is returned, otherwise `False` is returned (for instance when a collation with the same name is already loaded).

See also: UnRegisterCollation (2113), UnRegisterCollations (2113), LoadCollation (2109)

79.4.19 ToUCS4

Synopsis: Encode unicode UTF16 surrogate pair to UCS4 character.

Declaration:
```
function ToUCS4(const AHighS: UnicodeChar; const ALowS: UnicodeChar)
                        : UCS4Char
```

Visibility: default

Description: `ToUCS4` converts set of unicode (surrogate pair) characters encoded in UTF16: `AHighS`, `ALowS` to a UCS4 encoded unicode character.

See also: FromUCS4 (2107), UnicodeIsHighSurrogate (2111), UnicodeIsLowSurrogate (2111), UnicodeIsSurrogatePair (2112)

79.4.20 UnicodeIsHighSurrogate

Synopsis: Check if a UTF16 character is the high character in a surrogate pair.

Declaration:
```
function UnicodeIsHighSurrogate(const AValue: UnicodeChar) : Boolean
```

Visibility: default

Description: `UnicodeIsHighSurrogate` checks whether `AValue` is a valid high character of a surrogate pair, and returns `True` if this is the case. It does this by checking whether the values are within the bounds for high characters in surrogate pairs.

See also: FromUCS4 (2107), ToUCS4 (2111), UnicodeIsHighSurrogate (2111), UnicodeIsLowSurrogate (2111), UnicodeIsSurrogatePair (2112)

79.4.21 UnicodeIsLowSurrogate

Synopsis: Check if a UTF16 character is the low character in a surrogate pair.

Declaration:
```
function UnicodeIsLowSurrogate(const AValue: UnicodeChar) : Boolean
```

Visibility: default

Description: `UnicodeIsHighSurrogate` checks whether `AValue` is a valid high character of a surrogate pair, and returns `True` if this is the case. It does this by checking whether the values are within the bounds for high characters in surrogate pairs.

See also: FromUCS4 (2107), ToUCS4 (2111), UnicodeIsHighSurrogate (2111), UnicodeIsSurrogatePair (2112)

79.4.22 UnicodeIsSurrogatePair

Synopsis: Check if a pair of UTF16 encoded characters is a valid surrogate pair.

Declaration:
```
function UnicodeIsSurrogatePair(const AHighSurrogate: UnicodeChar;
                                const ALowSurrogate: UnicodeChar)
                                 : Boolean
```

Visibility: default

Description: `UnicodeIsSurrogatePair` checks whether `AHighSurrogate`,`ALowSurrogate` constitute a valid surrogate pair, and returns `True` if this is the case. It does this by checking whether the values are within the bounds for high and low surrogate pairs.

See also: FromUCS4 (2107), ToUCS4 (2111), UnicodeIsHighSurrogate (2111), UnicodeIsLowSurrogate (2111)

79.4.23 UnicodeToLower

Synopsis: Transform unicode string to lowercase.

Declaration:
```
function UnicodeToLower(const AString: UnicodeString;
                        const AIgnoreInvalidSequence: Boolean;
                        out AResultString: UnicodeString) : Integer
```

Visibility: default

Description: `UnicodeToLower` transforms a UTF16 unicode string `AString` to its lowercase equivalent and returns this in `AResultString`. If the transformation was successful, then the function returns `0`. A nonzero return value means an error occurred. `AResultString` will remain untouched in that case.

If a character in `AString` cannot be found in the unicode data tables, an error will be reported, unless `AIgnoreInvalidSequence` is set to `True`, in which case the character will be copied as-is to the output.

Unicode collation data can be loaded using RegisterCollation (2110) or LoadCollation (2109)

Errors: On error, a nonzero value will be returned.

See also: UnicodeToUpper (2112), RegisterCollation (2110), LoadCollation (2109)

79.4.24 UnicodeToUpper

Synopsis: Transform unicode string to uppercase.

Declaration:
```
function UnicodeToUpper(const AString: UnicodeString;
                        const AIgnoreInvalidSequence: Boolean;
                        out AResultString: UnicodeString) : Integer
```

Visibility: default

Description: `UnicodeToUpper` transforms a UTF16 unicode string `AString` to its uppercase equivalent and returns this in `AResultString`. If the transformation was successful, then the function returns `0`. A nonzero return value means an error occurred. `AResultString` will remain untouched in that case.

If a character in `AString` cannot be found in the unicode data tables, an error will be reported, unless `AIgnoreInvalidSequence` is set to `True`, in which case the character will be copied as-is to the output.

Unicode collation data can be loaded using RegisterCollation (2110) or LoadCollation (2109)

Errors: On error, a nonzero value will be returned.

See also: UnicodeToLower (2112), RegisterCollation (2110), LoadCollation (2109)

79.4.25 UnregisterCollation

Synopsis: Remove a collation from the list of known collections.

Declaration: `function UnregisterCollation(AName: UnicodeString) : Boolean`

Visibility: default

Description: `UnRegisterCollation` removes a collation `AName` from the list of known collations. It returns `True` if the collation was found and successfully removed.

Errors: If the collation was not found, `False` is returned.

See also: RegisterCollation (2110), UnRegisterCollations (2113), LoadCollation (2109)

79.4.26 UnregisterCollations

Synopsis: Unregister all collations.

Declaration: `procedure UnregisterCollations(const AFreeDynamicCollations: Boolean)`

Visibility: default

Description: `UnregisterCollations` unregisters all known collations. If `AFreeDynamicCollations` is `True`, then dynamic collations will be removed from memory using FreeCollation (2107). This must normally be set to `true`.

See also: RegisterCollation (2110), UnRegisterCollation (2113), LoadCollation (2109), FreeCollation (2107)

79.5 TCollationTable

```
TCollationTable = record
private
  FItems : TCollationTableItemArray
  ;
  FCount : Integer;
  function GetCapacity : Integer;
  function
  GetCount : Integer;
  function GetItem(const AIndex: Integer) : PCollationTableItem
  ;
  procedure Grow;
  procedure ClearItem(AItem: PCollationTableItem
  );
  AddAlias;
public
  class function NormalizeName(AName: UnicodeString
  ) : UnicodeString
                              ;  Static;
  procedure
  Clear;
```

```
  IndexOf;
  Find;
  function Add(ACollation: PUCA_DataBook
  ) : Integer;
  function Remove(AIndex: Integer) : PUCA_DataBook;
  property Item[AIndex: Integer]: PCollationTableItem; default;
  property Count : Integer;
  property Capacity : Integer;
end
```

79.5.1 Method overview

Page	Method	Description
2115	Add	
2114	Clear	
2114	Find	
2114	IndexOf	
2114	NormalizeName	
2115	Remove	

79.5.2 Property overview

Page	Properties	Access	Description
2115	Capacity	r	
2115	Count	r	
2115	Item	r	

79.5.3 TCollationTable.NormalizeName

Declaration: `class function NormalizeName(AName: UnicodeString) : UnicodeString ; Static`

Visibility: public

79.5.4 TCollationTable.Clear

Declaration: `procedure Clear`

Visibility: public

79.5.5 TCollationTable.IndexOf

Declaration:
```
function IndexOf(AName: UnicodeString) : Integer;  Overload
function IndexOf(ACollation: PUCA_DataBook) : Integer;  Overload
```

Visibility: public

79.5.6 TCollationTable.Find

Declaration:
```
function Find(AName: UnicodeString) : PCollationTableItem;  Overload
function Find(ACollation: PUCA_DataBook) : PCollationTableItem
            ;  Overload
```

Visibility: public

79.5.7 TCollationTable.Add

Declaration: `function Add(ACollation: PUCA_DataBook) : Integer`

Visibility: public

79.5.8 TCollationTable.Remove

Declaration: `function Remove(AIndex: Integer) : PUCA_DataBook`

Visibility: public

79.5.9 TCollationTable.Item

Declaration: `Property Item[AIndex: Integer]: PCollationTableItem; default`

Visibility: public

Access: Read

79.5.10 TCollationTable.Count

Declaration: `Property Count : Integer`

Visibility: public

Access: Read

79.5.11 TCollationTable.Capacity

Declaration: `Property Capacity : Integer`

Visibility: public

Access: Read

79.6 TUCA_DataBook

```
TUCA_DataBook = record
public
  Base : PUCA_DataBook;
  Version :
  TCollationVersion;
  CollationName : TCollationName;
  VariableWeight
  : TUCA_VariableKind;
  Backwards : Array[0..3] of Boolean;
  BMP_Table1
  : PByte;
  BMP_Table2 : PUInt24;
```

```
  OBMP_Table1 : PWord;
  OBMP_Table2
  : PUInt24;
  PropCount : Integer;
  Props : PUCA_PropItemRec;
  VariableLowLimit
  : Word;
  VariableHighLimit : Word;
  NoNormalization : Boolean;
  ComparisonStrength : Byte;
  Dynamic : Boolean;
  function IsVariable
  (const AWeight: PUCA_PropWeights) : Boolean;
end
```

TUCA_DataBook describes a Unicode Collation Algorithm data set. data sets can be registered using the RegisterCollation (2110) function or loaded from file using LoadCollation (2109). A collation data book must be specified when comparing unicode strings.

79.6.1 Method overview

Page	Method	Description
2116	IsVariable	Check if a weight is a variable weight.

79.6.2 TUCA_DataBook.IsVariable

Synopsis: Check if a weight is a variable weight.

Declaration: `function IsVariable(const AWeight: PUCA_PropWeights) : Boolean`

Visibility: public

Description: `IsVariable` checks whether `AWeight` is between the `VariableLowLimit` and `VariableHighLimit` limits.

79.7 TUCA_PropItemContextRec

```
TUCA_PropItemContextRec = packed record
public
  CodePointCount :
  Byte;
  WeightCount : Byte;
  function GetCodePoints : PUInt24;
  function GetWeights : PUCA_PropWeights;
end
```

This is an internal structure which should not be used directly, the actual structure is subject to change.

79.7.1 Method overview

Page	Method	Description
2117	GetCodePoints	get the address of actual code points.
2117	GetWeights	Get the Address of actual weights.

79.7.2 TUCA_PropItemContextRec.GetCodePoints

Synopsis: get the address of actual code points.

Declaration: `function GetCodePoints : PUInt24`

Visibility: public

79.7.3 TUCA_PropItemContextRec.GetWeights

Synopsis: Get the Address of actual weights.

Declaration: `function GetWeights : PUCA_PropWeights`

Visibility: public

79.8 TUCA_PropItemContextTreeNodeRec

```
TUCA_PropItemContextTreeNodeRec = packed record
public
  Left : Word
  ;
  Right : Word;
  Data : TUCA_PropItemContextRec;
  function GetLeftNode
  : PUCA_PropItemContextTreeNodeRec;
  function GetRightNode : PUCA_PropItemContextTreeNodeRec
  ;
end
```

This is an internal structure for the tree which should not be used directly, the actual structure is subject to change.

79.8.1 Method overview

Page	Method	Description
2117	GetLeftNode	Access to left tree node data.
2118	GetRightNode	Access to right tree node data.

79.8.2 TUCA_PropItemContextTreeNodeRec.GetLeftNode

Synopsis: Access to left tree node data.

Declaration: `function GetLeftNode : PUCA_PropItemContextTreeNodeRec`

Visibility: public

79.8.3 TUCA_PropItemContextTreeNodeRec.GetRightNode

Synopsis: Access to right tree node data.

Declaration: `function GetRightNode : PUCA_PropItemContextTreeNodeRec`

Visibility: public

79.9 TUCA_PropItemContextTreeRec

```
TUCA_PropItemContextTreeRec = packed record
public
  Size : UInt24
  ;
  function GetData : PUCA_PropItemContextTreeNodeRec;
  property
  Data : PUCA_PropItemContextTreeNodeRec;
  function Find(const AChars
  : PUInt24; const ACharCount: Integer;
               out ANode: PUCA_PropItemContextTreeNodeRec
  ) : Boolean;
end
```

This is an internal tree structure for storing unicode collation data which should not be used directly, the actual structure is subject to change.

79.9.1 Method overview

Page	Method	Description
2118	Find	Find data for encoded character.
2118	GetData	Access to tree data (getter for Data property).

79.9.2 Property overview

Page	Properties	Access	Description
2119	Data	r	Read-only access to tree data.

79.9.3 TUCA_PropItemContextTreeRec.GetData

Synopsis: Access to tree data (getter for Data property).

Declaration: `function GetData : PUCA_PropItemContextTreeNodeRec`

Visibility: public

79.9.4 TUCA_PropItemContextTreeRec.Find

Synopsis: Find data for encoded character.

Declaration:
```
function Find(const AChars: PUInt24; const ACharCount: Integer;
                  out ANode: PUCA_PropItemContextTreeNodeRec) : Boolean
```

Visibility: public

Description: `Find` searches the tree for the collation data for the character encoded in `AChars` (`ACharCount`). It returns true if the data was found, false if not. A pointer to the collation data is returned in `ANode`.

79.9.5 TUCA_PropItemContextTreeRec.Data

Synopsis: Read-only access to tree data.

Declaration: `Property Data : PUCA_PropItemContextTreeNodeRec`

Visibility: public

Access: Read

79.10 TUCA_PropItemRec

```
TUCA_PropItemRec = packed record
private
  FLAG_VALID = 0;
  FLAG_CODEPOINT
  = 1;
  FLAG_CONTEXTUAL = 2;
  FLAG_DELETION = 3;
  FLAG_COMPRESS_WEIGHT_1
  = 6;
  FLAG_COMPRESS_WEIGHT_2 = 7;
  function GetCodePoint : UInt24
  ;
public
  WeightLength : Byte;
  ChildCount : Byte;
  Size : Word
  ;
  Flags : Byte;
  function HasCodePoint : Boolean;
  property CodePoint
  : UInt24;
  function IsValid : Boolean;
  procedure GetWeightArray
  (ADest: PUCA_PropWeights);
  function GetSelfOnlySize : Cardinal;
  function GetContextual : Boolean;
  property Contextual : Boolean
  ;
  function GetContext : PUCA_PropItemContextTreeRec;
  function
  IsDeleted : Boolean;
  function IsWeightCompress_1 : Boolean;
  function
  IsWeightCompress_2 : Boolean;
end
```

`TUCA_PropItemRec` encodes 1 entry from the Unicode Collation data in an encoded form.

79.10.1 Method overview

Page	Method	Description
2121	GetContext	Access to context data.
2121	GetContextual	Check if the contextual bit is set in the flags (getter for Contextual).
2120	GetSelfOnlySize	Size of this item data (in bytes).
2120	GetWeightArray	Return an array of weights.
2120	HasCodePoint	Check flags whether a codepoint is present.
2121	IsDeleted	Check flags if deleted bit is set.
2120	IsValid	Check flags for validity.
2121	IsWeightCompress_1	Check whether weight compression flag 1 is set.
2121	IsWeightCompress_2	Check whether weight compression flag 2 is set.

79.10.2 Property overview

Page	Properties	Access	Description
2121	CodePoint	r	Get the codepoint.
2122	Contextual	r	Check if the contextual bit is set in the flags.

79.10.3 TUCA_PropItemRec.HasCodePoint

Synopsis: Check flags whether a codepoint is present.

Declaration: `function HasCodePoint : Boolean`

Visibility: public

79.10.4 TUCA_PropItemRec.IsValid

Synopsis: Check flags for validity.

Declaration: `function IsValid : Boolean`

Visibility: public

79.10.5 TUCA_PropItemRec.GetWeightArray

Synopsis: Return an array of weights.

Declaration: `procedure GetWeightArray(ADest: PUCA_PropWeights)`

Visibility: public

Description: `GetWeightArray` returns an array with the weights in `ADest`. `ADest` must point to enough room for `WeightLength` weights.

79.10.6 TUCA_PropItemRec.GetSelfOnlySize

Synopsis: Size of this item data (in bytes).

Declaration: `function GetSelfOnlySize : Cardinal`

Visibility: public

Description: Return the size of the item's data and properties.

79.10.7 TUCA_PropItemRec.GetContextual

Synopsis: Check if the contextual bit is set in the flags (getter for Contextual).

Declaration: `function GetContextual : Boolean`

Visibility: public

79.10.8 TUCA_PropItemRec.GetContext

Synopsis: Access to context data.

Declaration: `function GetContext : PUCA_PropItemContextTreeRec`

Visibility: public

Description: `GetContext` returns a pointer to the context data. It is `Nil` if the context flag is not set.

79.10.9 TUCA_PropItemRec.IsDeleted

Synopsis: Check flags if deleted bit is set.

Declaration: `function IsDeleted : Boolean`

Visibility: public

79.10.10 TUCA_PropItemRec.IsWeightCompress_1

Synopsis: Check whether weight compression flag 1 is set.

Declaration: `function IsWeightCompress_1 : Boolean`

Visibility: public

79.10.11 TUCA_PropItemRec.IsWeightCompress_2

Synopsis: Check whether weight compression flag 2 is set.

Declaration: `function IsWeightCompress_2 : Boolean`

Visibility: public

79.10.12 TUCA_PropItemRec.CodePoint

Synopsis: Get the codepoint.

Declaration: `Property CodePoint : UInt24`

Visibility: public

Access: Read

Description: Access to codepoint if `HasCodePoint` returns true. If `HasCodePoint` returns false, an exception will be raised.

79.10.13 TUCA_PropItemRec.Contextual

Synopsis: Check if the contextual bit is set in the flags.

Declaration: `Property Contextual : Boolean`

Visibility: public

Access: Read

79.11 TUC_Prop

```
TUC_Prop = packed record
private
  function GetCategory : Byte;
  procedure SetCategory(AValue: Byte);
  function GetWhiteSpace : Boolean
  ;
  procedure SetWhiteSpace(AValue: Boolean);
  function GetHangulSyllable
  : Boolean;
  procedure SetHangulSyllable(AValue: Boolean);
  function
  GetNumericValue : Double;
public
  CategoryData : Byte;
  CCC : Byte
  ;
  NumericIndex : Byte;
  SimpleUpperCase : UInt24;
  SimpleLowerCase
  : UInt24;
  DecompositionID : SmallInt;
  property Category : Byte
  ;
  property WhiteSpace : Boolean;
  property HangulSyllable : Boolean
  ;
  property NumericValue : Double;
end
```

`TUC_Prop` describes the collation characteristics of a unicode character. It is an internal structure which should not be used directly, the actual structure is subject to change.

79.11.1 Property overview

Page	Properties	Access	Description
2123	Category	rw	Get the category.
2123	HangulSyllable	rw	Is the character a hangul syllable.
2123	NumericValue	r	Numeric value.
2123	WhiteSpace	rw	Is the character considered whitespace.

79.11.2 TUC_Prop.Category

Synopsis: Get the category.

Declaration: `Property Category : Byte`

Visibility: public

Access: Read,Write

Description: `CategoryData` provides access to the category part of CategoryData (encoded).

79.11.3 TUC_Prop.WhiteSpace

Synopsis: Is the character considered whitespace.

Declaration: `Property WhiteSpace : Boolean`

Visibility: public

Access: Read,Write

Description: `Whitespace` provides easy access to the Whitespace part of CategoryData (encoded).

79.11.4 TUC_Prop.HangulSyllable

Synopsis: Is the character a hangul syllable.

Declaration: `Property HangulSyllable : Boolean`

Visibility: public

Access: Read,Write

Description: `Whitespace` provides easy access to the HangulSyllable part of CategoryData (encoded).

79.11.5 TUC_Prop.NumericValue

Synopsis: Numeric value.

Declaration: `Property NumericValue : Double`

Visibility: public

Access: Read

Description: `NumericValue` uses numericalindex to get the numerical value.

79.12 TUInt24Rec

```
TUInt24Rec = packed record
public
  byte2 : Byte;
  byte1 : Byte;
  byte0 : Byte;
  TUInt24Rec.class operator implicit(a: TUInt24Rec
  ) : Cardinal;
```

```
TUInt24Rec.class operator implicit(a: TUInt24Rec)
: LongInt;
TUInt24Rec.class operator implicit(a: TUInt24Rec) :
Word;
TUInt24Rec.class operator implicit(a: TUInt24Rec) : Byte
;
TUInt24Rec.class operator implicit(a: Cardinal) : TUInt24Rec;
TUInt24Rec.class operator equal(a: TUInt24Rec; b: TUInt24Rec)
: Boolean;
TUInt24Rec.class operator equal(a: TUInt24Rec; b: Cardinal
) : Boolean;
TUInt24Rec.class operator equal(a: Cardinal; b: TUInt24Rec
) : Boolean;
TUInt24Rec.class operator equal(a: TUInt24Rec; b: LongInt
) : Boolean;
TUInt24Rec.class operator equal(a: LongInt; b: TUInt24Rec
) : Boolean;
TUInt24Rec.class operator equal(a: TUInt24Rec; b: Word
) : Boolean;
TUInt24Rec.class operator equal(a: Word; b: TUInt24Rec
) : Boolean;
TUInt24Rec.class operator equal(a: TUInt24Rec; b: Byte
) : Boolean;
TUInt24Rec.class operator equal(a: Byte; b: TUInt24Rec
) : Boolean;
TUInt24Rec.class operator notequal(a: TUInt24Rec; b
: TUInt24Rec)
                                  : Boolean;
TUInt24Rec
.class operator notequal(a: TUInt24Rec; b: Cardinal) : Boolean;
TUInt24Rec.class operator notequal(a: Cardinal; b: TUInt24Rec) :
Boolean;
TUInt24Rec.class operator greaterthan(a: TUInt24Rec; b
: TUInt24Rec)
                                    : Boolean;
TUInt24Rec.class operator greaterthan(a: TUInt24Rec; b: Cardinal
)
                                    : Boolean;
TUInt24Rec
.class operator greaterthan(a: Cardinal; b: TUInt24Rec)
: Boolean;
TUInt24Rec.class operator
greaterthanorequal(a: TUInt24Rec;
b: TUInt24Rec) : Boolean;
TUInt24Rec.class operator
greaterthanorequal(a: TUInt24Rec; b: Cardinal)
: Boolean;
TUInt24Rec.class operator
greaterthanorequal(a: Cardinal; b: TUInt24Rec)
: Boolean;
TUInt24Rec.class operator
lessthan(a: TUInt24Rec; b: TUInt24Rec)
: Boolean;
TUInt24Rec.class operator lessthan(a: TUInt24Rec
```

```
  ; b: Cardinal) : Boolean;
  TUInt24Rec.class operator lessthan(a:
  Cardinal; b: TUInt24Rec) : Boolean;
  TUInt24Rec.class operator lessthanorequal
  (a: TUInt24Rec; b: TUInt24Rec)
  : Boolean;
  TUInt24Rec.class operator lessthanorequal(a
  : TUInt24Rec; b: Cardinal)
  : Boolean;
  TUInt24Rec.class operator lessthanorequal(a: Cardinal
  ; b: TUInt24Rec)
                                              : Boolean
  ;
end
```

Unicode data exists mostly of 24-bit data (3 bytes). This type is meant to deal efficiently with this data. it has members to split out the data in bytes, and functions to query the various properties stored in the data.

79.12.1 Method overview

Page	Method	Description
2130	equal(Byte,TUInt24Rec):Boolean	Check whether a `TUInt24Rec` value equals a byte value.
2129	equal(Cardinal,TUInt24Rec):Boolean	Check whether a `TUInt24Rec` value equals a cardinal value.
2130	equal(LongInt,TUInt24Rec):Boolean	Check whether a `TUInt24Rec` value equals a longint value.
2130	equal(TUInt24Rec,Byte):Boolean	Check whether a `TUInt24Rec` value equals a byte value.
2129	equal(TUInt24Rec,Cardinal):Boolean	Check if cardinal and `TUInt24Rec` are equal.
2129	equal(TUInt24Rec,LongInt):Boolean	Check whether a `TUInt24Rec` value equals a longint value.
2129	equal(TUInt24Rec,TUInt24Rec):Boolean	Determine equality of 2 `TUInt24Rec` records.
2130	equal(TUInt24Rec,Word):Boolean	Check whether a `TUInt24Rec` value equals a word value.
2130	equal(Word,TUInt24Rec):Boolean	Check whether a `TUInt24Rec` value equals a word value.
2131	greaterthan(Cardinal,TUInt24Rec):Boolean	Check whether a cardinal value is greater than a `TUInt24Rec` value.
2131	greaterthan(TUInt24Rec,Cardinal):Boolean	Check whether a `TUInt24Rec` value is greater than a cardinal value.
2131	greaterthan(TUInt24Rec,TUInt24Rec):Boolean	Check whether a `TUInt24Rec` value is greater than another `TUInt24Rec` value.
2132	greaterthanorequal(Cardinal,TUInt24Rec):Boolean	Check whether a cardinal value is greater than or equal to a `TUInt24Rec` value.
2132	greaterthanorequal(TUInt24Rec,Cardinal):Boolean	Check whether a `TUInt24Rec` value is greater than or equal to a cardinal value.
2132	greaterthanorequal(TUInt24Rec,TUInt24Rec):Boolean	Check whether a `TUInt24Rec` value is greater than or equal to a cardinal value.
2129	implicit(Cardinal):TUInt24Rec	Assign `TUInt24Rec`from Cardinal.
2128	implicit(TUInt24Rec):Byte	Assign `TUInt24Rec` to byte.
2128	implicit(TUInt24Rec):Cardinal	Assign `TUInt24Rec` to cardinal.
2128	implicit(TUInt24Rec):LongInt	Assign `TUInt24Rec` to longint.
2128	implicit(TUInt24Rec):Word	Assign `TUInt24Rec` to word.
2133	lessthan(Cardinal,TUInt24Rec):Boolean	Check whether a cardinal value is less than a `TUInt24Rec` value.
2132	lessthan(TUInt24Rec,Cardinal):Boolean	Check whether a

79.12.2 TUInt24Rec.implicit(TUInt24Rec):Cardinal

Synopsis: Assign `TUInt24Rec` to cardinal.

Declaration: `TUInt24Rec.class operator implicit(a: TUInt24Rec) : Cardinal`

Visibility: public

Description: Assign to cardinal, byte0 to MSB and so on.

See also: TUInt24Rec.implicit(TUInt24Rec):LongInt (2128), TUInt24Rec.implicit(TUInt24Rec):Word (2128), TUInt24Rec.implicit(TUInt24Rec):Byte (2128), TUInt24Rec.implicit(Cardinal):TUInt24Rec (2129), TUInt24Rec.implicit(Longint):TUInt24Rec (2125), TUInt24Rec.implicit(Word):TUInt24Rec (2125), TUInt24Rec.implicit(Byte):TUInt24Rec (2125)

79.12.3 TUInt24Rec.implicit(TUInt24Rec):LongInt

Synopsis: Assign `TUInt24Rec` to longint.

Declaration: `TUInt24Rec.class operator implicit(a: TUInt24Rec) : LongInt`

Visibility: public

Description: Assign to cardinal, byte0 to MSB and so on.

See also: TUInt24Rec.implicit(TUInt24Rec):Cardinal (2128), TUInt24Rec.implicit(TUInt24Rec):Word (2128), TUInt24Rec.implicit(TUInt24Rec):Byte (2128)

79.12.4 TUInt24Rec.implicit(TUInt24Rec):Word

Synopsis: Assign `TUInt24Rec` to word.

Declaration: `TUInt24Rec.class operator implicit(a: TUInt24Rec) : Word`

Visibility: public

Description: Assign to word, byte0 to MSB, byte1 to MSB.

Errors: If the value is too big (>$FFFF) to be assigned, an overflow error occurs.

See also: TUInt24Rec.implicit(TUInt24Rec):Cardinal (2128), TUInt24Rec.implicit(TUInt24Rec):Longint (2128), TUInt24Rec.implicit(TUInt24Rec):Byte (2128), TUInt24Rec.implicit(Cardinal):TUInt24Rec (2129), TUInt24Rec.implicit(Longint):TUInt24Rec (2125), TUInt24Rec.implicit(Word):TUInt24Rec (2125), TUInt24Rec.implicit(Byte):TUInt24Rec (2125)

79.12.5 TUInt24Rec.implicit(TUInt24Rec):Byte

Synopsis: Assign `TUInt24Rec` to byte.

Declaration: `TUInt24Rec.class operator implicit(a: TUInt24Rec) : Byte`

Visibility: public

Description: Assign to byte, byte0 is assigned.

Errors: If the value is too big (>$FF) to be assigned, an overflow error occurs.

See also: TUInt24Rec.implicit(TUInt24Rec):Cardinal (2128), TUInt24Rec.implicit(TUInt24Rec):Longint (2128), TUInt24Rec.implicit(TUInt24Rec):Word (2128), TUInt24Rec.implicit(Cardinal):TUInt24Rec (2129), TUInt24Rec.implicit(Longint):TUInt24Rec (2125), TUInt24Rec.implicit(Word):TUInt24Rec (2125), TUInt24Rec.implicit(Byte):TUInt24Rec (2125)

79.12.6 TUInt24Rec.implicit(Cardinal):TUInt24Rec

Synopsis: Assign `TUInt24Rec`from Cardinal.

Declaration: `TUInt24Rec.class operator implicit(a: Cardinal) : TUInt24Rec`

Visibility: public

Description: Assign from cardinal, byte0 to MSB.

Errors: If the value is too big (>$FFFFFF) to be assigned, an overflow error occurs.

See also: TUInt24Rec.implicit(TUInt24Rec):Longint (2128), TUInt24Rec.implicit(TUInt24Rec):Word (2128), TUInt24Rec.implicit(TUInt24Rec):Byte (2128), TUInt24Rec.implicit(Longint):TUInt24Rec (2125), TUInt24Rec.implicit(Word):TUInt24Rec (2125), TUInt24Rec.implicit(Byte):TUInt24Rec (2125)

79.12.7 TUInt24Rec.equal(TUInt24Rec,TUInt24Rec):Boolean

Synopsis: Determine equality of 2 `TUInt24Rec` records.

Declaration: `TUInt24Rec.class operator equal(a: TUInt24Rec; b: TUInt24Rec) : Boolean`

Visibility: public

Description: The 2 records are considered equal if the 3 bytes are equal.

79.12.8 TUInt24Rec.equal(TUInt24Rec,Cardinal):Boolean

Synopsis: Check if cardinal and `TUInt24Rec` are equal.

Declaration: `TUInt24Rec.class operator equal(a: TUInt24Rec; b: Cardinal) : Boolean`

Visibility: public

Description: The cardinal `b` is considered equal to `a` if the fourth byte (LSB) is zero, and the first three bytes equal `byte0`, `byte1`, and `byte2`.

79.12.9 TUInt24Rec.equal(Cardinal,TUInt24Rec):Boolean

Synopsis: Check whether a `TUInt24Rec` value equals a cardinal value.

Declaration: `TUInt24Rec.class operator equal(a: Cardinal; b: TUInt24Rec) : Boolean`

Visibility: public

Description: The cardinal `a` is considered equal to `b` if the fourth byte (LSB) is zero, and the first three bytes equal `byte0`, `byte1`, and `byte2`.

79.12.10 TUInt24Rec.equal(TUInt24Rec,LongInt):Boolean

Synopsis: Check whether a `TUInt24Rec` value equals a longint value.

Declaration: `TUInt24Rec.class operator equal(a: TUInt24Rec; b: LongInt) : Boolean`

Visibility: public

Description: The longint `b` is considered equal to `a` if the fourth byte (LSB) is zero, and the first three bytes equal `byte0`, `byte1`, and `byte2`.

79.12.11 TUInt24Rec.equal(LongInt,TUInt24Rec):Boolean

Synopsis: Check whether a `TUInt24Rec` value equals a longint value.

Declaration:
```
TUInt24Rec.class operator equal(a: LongInt; b: TUInt24Rec) : Boolean
```

Visibility: public

Description: The longint `a` is considered equal to `b` if the fourth byte (LSB) is zero, and the first three bytes equal `byte0`, `byte1`, and `byte2`.

79.12.12 TUInt24Rec.equal(TUInt24Rec,Word):Boolean

Synopsis: Check whether a `TUInt24Rec` value equals a word value.

Declaration:
```
TUInt24Rec.class operator equal(a: TUInt24Rec; b: Word) : Boolean
```

Visibility: public

Description: The word `b` is considered equal to `a` if its 2 bytes equal `byte0`, `byte1` and `byte2` is zero.

79.12.13 TUInt24Rec.equal(Word,TUInt24Rec):Boolean

Synopsis: Check whether a `TUInt24Rec` value equals a word value.

Declaration:
```
TUInt24Rec.class operator equal(a: Word; b: TUInt24Rec) : Boolean
```

Visibility: public

Description: The word `a` is considered equal to `b` if its 2 bytes equal `byte0`, `byte1` and `byte2` is zero.

79.12.14 TUInt24Rec.equal(TUInt24Rec,Byte):Boolean

Synopsis: Check whether a `TUInt24Rec` value equals a byte value.

Declaration:
```
TUInt24Rec.class operator equal(a: TUInt24Rec; b: Byte) : Boolean
```

Visibility: public

Description: The byte `b` is considered equal to `a` if it equals `byte0` and `byte1` and `byte1` are zero.

79.12.15 TUInt24Rec.equal(Byte,TUInt24Rec):Boolean

Synopsis: Check whether a `TUInt24Rec` value equals a byte value.

Declaration:
```
TUInt24Rec.class operator equal(a: Byte; b: TUInt24Rec) : Boolean
```

Visibility: public

Description: The byte `b` is considered equal to `a` if it equals `byte0` and `byte1` and `byte1` are zero.

79.12.16 TUInt24Rec.notequal(TUInt24Rec,TUInt24Rec):Boolean

Synopsis: Check whether 2 `TUInt24Rec` values differ.

Declaration:
```
TUInt24Rec.class operator notequal(a: TUInt24Rec; b: TUInt24Rec)
                                        : Boolean
```

Visibility: public

Description: The comparison is done by comparing `byte0`, `byte1` and `byte2`.

79.12.17 TUInt24Rec.notequal(TUInt24Rec,Cardinal):Boolean

Synopsis: Check whether a `TUInt24Rec` value differs from a cardinal value.

Declaration:
```
TUInt24Rec.class operator notequal(a: TUInt24Rec; b: Cardinal) : Boolean
```

Visibility: public

Description: The cardinal `a` is considered not equal to `b` if the fourth byte (LSB) is nonzero, or one of the first three bytes differ from `byte0`, `byte1`, and `byte2`.

79.12.18 TUInt24Rec.notequal(Cardinal,TUInt24Rec):Boolean

Synopsis: Check whether a `TUInt24Rec` value differs from a cardinal value.

Declaration:
```
TUInt24Rec.class operator notequal(a: Cardinal; b: TUInt24Rec) : Boolean
```

Visibility: public

Description: The cardinal `a` is considered not equal to `b` if the fourth byte (LSB) is nonzero, or one of the first three bytes differ from `byte0`, `byte1`, and `byte2`.

79.12.19 TUInt24Rec.greaterthan(TUInt24Rec,TUInt24Rec):Boolean

Synopsis: Check whether a `TUInt24Rec` value is greater than another `TUInt24Rec` value.

Declaration:
```
TUInt24Rec.class operator greaterthan(a: TUInt24Rec; b: TUInt24Rec)
                                      : Boolean
```

Visibility: public

Description: The comparison is done by comparing 3 bytes `byte0`, `byte1` and `byte2`

79.12.20 TUInt24Rec.greaterthan(TUInt24Rec,Cardinal):Boolean

Synopsis: Check whether a `TUInt24Rec` value is greater than a cardinal value.

Declaration:
```
TUInt24Rec.class operator greaterthan(a: TUInt24Rec; b: Cardinal)
                                      : Boolean
```

Visibility: public

Description: The comparison is done by comparing 3 first bytes of the cardinal value with bytes `byte0`, `byte1` and `byte2`.

79.12.21 TUInt24Rec.greaterthan(Cardinal,TUInt24Rec):Boolean

Synopsis: Check whether a cardinal value is greater than a `TUInt24Rec` value.

Declaration:
```
TUInt24Rec.class operator greaterthan(a: Cardinal; b: TUInt24Rec)
                                      : Boolean
```

Visibility: public

Description: The comparison is done by comparing 3 first bytes of the cardinal value with bytes `byte0`, `byte1` and `byte2`.

79.12.22 TUInt24Rec.greaterthanorequal(TUInt24Rec,TUInt24Rec):Boolean

Synopsis: Check whether a `TUInt24Rec` value is greater than or equal to a cardinal value.

Declaration:
```
TUInt24Rec.class operator greaterthanorequal(a: TUInt24Rec;
                                             b: TUInt24Rec) : Boolean
```

Visibility: public

Description: The comparison is done by comparing 3 first bytes of the cardinal value with bytes `byte0`, `byte1` and `byte2`.

79.12.23 TUInt24Rec.greaterthanorequal(TUInt24Rec,Cardinal):Boolean

Synopsis: Check whether a `TUInt24Rec` value is greater than or equal to a cardinal value.

Declaration:
```
TUInt24Rec.class operator greaterthanorequal(a: TUInt24Rec; b: Cardinal)
                                             : Boolean
```

Visibility: public

Description: The comparison is done by comparing 3 first bytes of the cardinal value with bytes `byte0`, `byte1` and `byte2`.

79.12.24 TUInt24Rec.greaterthanorequal(Cardinal,TUInt24Rec):Boolean

Synopsis: Check whether a cardinal value is greater than or equal to a `TUInt24Rec` value.

Declaration:
```
TUInt24Rec.class operator greaterthanorequal(a: Cardinal; b: TUInt24Rec)
                                             : Boolean
```

Visibility: public

Description: The comparison is done by comparing 3 first bytes of the cardinal value with bytes `byte0`, `byte1` and `byte2`.

79.12.25 TUInt24Rec.lessthan(TUInt24Rec,TUInt24Rec):Boolean

Synopsis: Check whether a `TUInt24Rec` value is less than another `TUInt24Rec` value.

Declaration:
```
TUInt24Rec.class operator lessthan(a: TUInt24Rec; b: TUInt24Rec)
                                   : Boolean
```

Visibility: public

Description: The comparison is done by comparing the bytes `byte0`, `byte1` and `byte2`.

79.12.26 TUInt24Rec.lessthan(TUInt24Rec,Cardinal):Boolean

Synopsis: Check whether a `TUInt24Rec` value is less than a cardinal value.

Declaration:
```
TUInt24Rec.class operator lessthan(a: TUInt24Rec; b: Cardinal) : Boolean
```

Visibility: public

Description: The comparison is done by comparing 3 first bytes of the cardinal value with bytes `byte0`, `byte1` and `byte2`.

79.12.27 TUInt24Rec.lessthan(Cardinal,TUInt24Rec):Boolean

Synopsis: Check whether a cardinal value is less than a `TUInt24Rec` value.

Declaration:
```
TUInt24Rec.class operator lessthan(a: Cardinal; b: TUInt24Rec) : Boolean
```

Visibility: public

Description: Check whether a cardinal value is less than a `TUInt24Rec` value.

79.12.28 TUInt24Rec.lessthanorequal(TUInt24Rec,TUInt24Rec):Boolean

Synopsis: Check whether a `TUInt24Rec` value is less than or equal to another `TUInt24Rec` value.

Declaration:
```
TUInt24Rec.class operator lessthanorequal(a: TUInt24Rec; b: TUInt24Rec)
                                                        : Boolean
```

Visibility: public

Description: The comparison is done by comparing the bytes `byte0`, `byte1` and `byte2`.

79.12.29 TUInt24Rec.lessthanorequal(TUInt24Rec,Cardinal):Boolean

Synopsis: Check whether a `TUInt24Rec` value is less than or equal to a cardinal value.

Declaration:
```
TUInt24Rec.class operator lessthanorequal(a: TUInt24Rec; b: Cardinal)
                                                        : Boolean
```

Visibility: public

Description: The comparison is done by comparing 3 first bytes of the cardinal value with bytes `byte0`, `byte1` and `byte2`.

79.12.30 TUInt24Rec.lessthanorequal(Cardinal,TUInt24Rec):Boolean

Synopsis: Check whether a cardinal value is less than or equal to a `TUInt24Rec` value.

Declaration:
```
TUInt24Rec.class operator lessthanorequal(a: Cardinal; b: TUInt24Rec)
                                                        : Boolean
```

Visibility: public

Description: The comparison is done by comparing 3 first bytes of the cardinal value with bytes `byte0`, `byte1` and `byte2`.

Chapter 80

Reference for unit 'unicodeducet'

80.1 Used units

Table 80.1: Used units by unit 'unicodeducet'

Name	Page
System	1362

80.2 Overview

The unicodeducet unit registers the root Unicode collation (DUCET). This collation is needed by all other collations, so any collation unit will include this file.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 81

Reference for unit 'Unix'

81.1 Used units

Table 81.1: Used units by unit 'Unix'

Name	Page
BaseUnix	146
System	1362
unixtype	2175

81.2 Constants, types and variables

81.2.1 Constants

`ARG_MAX = UnixType.ARG_MAX`

Maximum number of arguments to a program.

`fs_ext = $137d`

File system type (TStatFS (2150)): (ext) Extended.

`fs_ext2 = $ef53`

File system type (TStatFS (2150)): (ext2) Second extended.

`fs_iso = $9660`

File system type (TStatFS (2150)): ISO 9660.

`fs_minix = $137f`

File system type (TStatFS (2150)): Minix.

`fs_minix_30 = $138f`

File system type (TStatFS (2150)): Minix 3.0.

```
fs_minux_V2 = $2468
```

File system type (TStatFS (2150)): Minix V2.

```
fs_msdos = $4d44
```

File system type (TStatFS (2150)): MSDOS (FAT).

```
fs_nfs = $6969
```

File system type (TStatFS (2150)): NFS.

```
fs_old_ext2 = $ef51
```

File system type (TStatFS (2150)): (ext2) Old second extended.

```
fs_proc = $9fa0
```

File system type (TStatFS (2150)): PROC fs.

```
fs_xia = $012FD16D
```

File system type (TStatFS (2150)): XIA.

```
IOCtl_TCGETS = $5401
```

IOCTL call number: get Terminal Control settings.

```
LOCK_EX = 2
```

FpFLock (2159) Exclusive lock.

```
LOCK_NB = 4
```

FpFLock (2159) Non-blocking operation.

```
LOCK_SH = 1
```

FpFLock (2159) Shared lock.

```
LOCK_UN = 8
```

FpFLock (2159) unlock.

```
MAP_FAILED = baseunix.MAP_FAILED
```

Error return value for mmap: mmap operation failed.

```
MAP_FIXED = baseunix.MAP_FIXED
```

#rtl.baseunix.FpMMap (207) map type: Interpret addr exactly.

```
MAP_PRIVATE = baseunix.MAP_PRIVATE
```

#rtl.baseunix.FpMMap (207) map type: Changes are private.

```
MAP_SHARED = baseunix.MAP_SHARED
```

#rtl.baseunix.FpMMap (207) map type: Share changes.

```
MAP_TYPE = baseunix.MAP_TYPE
```

#rtl.baseunix.FpMMap (207) map type: Bitmask for type of mapping.

```
NAME_MAX = UnixType.NAME_MAX
```

Maximum filename length.

```
Open_Accmode = 3
```

Bitmask to determine access mode in open flags.

```
Open_Append = 2 shl 9
```

File open mode: Append to file.

```
Open_Creat = 1 shl 6
```

File open mode: Create if file does not yet exist.

```
Open_Direct = 4 shl 12
```

File open mode: Minimize caching effects.

```
Open_Directory = 2 shl 15
```

File open mode: File must be directory.

```
Open_Excl = 2 shl 6
```

File open mode: Open exclusively.

```
Open_LargeFile = 1 shl 15
```

File open mode: Open for 64-bit I/O.

```
Open_NDelay = Open_NonBlock
```

File open mode: Alias for Open_NonBlock (2138).

```
Open_NoCtty = 4 shl 6
```

File open mode: No TTY control.

```
Open_NoFollow = 4 shl 15
```

File open mode: Fail if file is symbolic link.

```
Open_NonBlock = 4 shl 9
```

File open mode: Open in non-blocking mode.

```
Open_RdOnly = 0
```

File open mode: Read only.

```
Open_RdWr = 2
```

File open mode: Read/Write.

```
Open_Sync = 1 shl 12
```

File open mode: Write to disc at once.

```
Open_Trunc = 1 shl 9
```

File open mode: Truncate file to length 0.

```
Open_WrOnly = 1
```

File open mode: Write only.

```
PATH_MAX = UnixType.PATH_MAX
```

Maximum pathname length.

```
PRIO_PGRP = UnixType.PRIO_PGRP
```

#rtl.baseunix.fpGetPriority (201) option: Get process group priority.

```
PRIO_PROCESS = UnixType.PRIO_PROCESS
```

#rtl.baseunix.fpGetPriority (201) option: Get process priority.

```
PRIO_USER = UnixType.PRIO_USER
```

#rtl.baseunix.fpGetPriority (201) option: Get user priority.

```
PROT_EXEC = baseunix.PROT_EXEC
```

#rtl.baseunix.FpMMap (207) memory access: page can be executed.

```
PROT_NONE = baseunix.PROT_NONE
```

#rtl.baseunix.FpMMap (207) memory access: page can not be accessed.

```
PROT_READ = baseunix.PROT_READ
```

#rtl.baseunix.FpMMap (207) memory access: page can be read.

```
PROT_WRITE = baseunix.PROT_WRITE
```

#rtl.baseunix.FpMMap (207) memory access: page can be written.

```
P_IN = 1
```

Input file descriptor of pipe pair.

```
P_OUT = 2
```

Output file descriptor of pipe pair.

```
SIG_MAXSIG = UnixType.SIG_MAXSIG
```

Maximum system signal number.

```
STAT_IFBLK = $6000
```

File (#rtl.baseunix.stat (240) record) mode: Block device.

```
STAT_IFCHR = $2000
```

File (#rtl.baseunix.stat (240) record) mode: Character device.

```
STAT_IFDIR = $4000
```

File (#rtl.baseunix.stat (240) record) mode: Directory.

```
STAT_IFIFO = $1000
```

File (#rtl.baseunix.stat (240) record) mode: FIFO.

```
STAT_IFLNK = $a000
```

File (#rtl.baseunix.stat (240) record) mode: Link.

```
STAT_IFMT = $f000
```

File (#rtl.baseunix.stat (240) record) mode: File type bit mask.

```
STAT_IFREG = $8000
```

File (#rtl.baseunix.stat (240) record) mode: Regular file.

```
STAT_IFSOCK = $c000
```

File (#rtl.baseunix.stat (240) record) mode: Socket.

```
STAT_IRGRP = STAT_IROTH shl 3
```

File (#rtl.baseunix.stat (240) record) mode: Group read permission.

```
STAT_IROTH = $4
```

File (#rtl.baseunix.stat (240) record) mode: Other read permission.

```
STAT_IRUSR = STAT_IROTH shl 6
```

File (#rtl.baseunix.stat (240) record) mode: Owner read permission.

```
STAT_IRWXG = STAT_IRWXO shl 3
```

File (#rtl.baseunix.stat (240) record) mode: Group permission bits mask.

```
STAT_IRWXO = $7
```

File (#rtl.baseunix.stat (240) record) mode: Other permission bits mask.

```
STAT_IRWXU = STAT_IRWXO shl 6
```

File (#rtl.baseunix.stat (240) record) mode: Owner permission bits mask.

```
STAT_ISGID = $0400
```

File (#rtl.baseunix.stat (240) record) mode: GID bit set.

```
STAT_ISUID = $0800
```

File (#rtl.baseunix.stat (240) record) mode: UID bit set.

```
STAT_ISVTX = $0200
```

File (#rtl.baseunix.stat (240) record) mode: Sticky bit set.

```
STAT_IWGRP = STAT_IWOTH shl 3
```

File (#rtl.baseunix.stat (240) record) mode: Group write permission.

```
STAT_IWOTH = $2
```

File (#rtl.baseunix.stat (240) record) mode: Other write permission.

```
STAT_IWUSR = STAT_IWOTH shl 6
```

File (#rtl.baseunix.stat (240) record) mode: Owner write permission.

```
STAT_IXGRP = STAT_IXOTH shl 3
```

File (#rtl.baseunix.stat (240) record) mode: Others execute permission.

```
STAT_IXOTH = $1
```

File (#rtl.baseunix.stat (240) record) mode: Others execute permission.

```
STAT_IXUSR = STAT_IXOTH shl 6
```

File (#rtl.baseunix.stat (240) record) mode: Others execute permission.

```
SYS_NMLN = UnixType.SYS_NMLN
```

Max system name length.

```
Wait_Any = - 1
```

#rtl.baseunix.fpWaitPID (236): Wait on any process.

```
Wait_Clone = $80000000
```

#rtl.baseunix.fpWaitPID (236): Wait on clone processes only.

```
Wait_MyPGRP = 0
```

#rtl.baseunix.fpWaitPID (236): Wait processes from current process group.

```
Wait_NoHang = 1
```

#rtl.baseunix.fpWaitPID (236): Do not wait.

```
Wait_UnTraced = 2
```

#rtl.baseunix.fpWaitPID (236): Also report stopped but untraced processes.

81.2.2 Types

```
cbool = UnixType.cbool
```

Boolean type.

```
cchar = UnixType.cchar
```

Alias for #rtl.UnixType.cchar (2177).

```
cdouble = UnixType.cdouble
```

Double precision real format.

```
cfloat = UnixType.cfloat
```

Floating-point real format.

```
cint = UnixType.cint
```

C type: integer (natural size).

```
cint16 = UnixType.cint16
```

C type: 16 bits sized, signed integer.

```
cint32 = UnixType.cint32
```

C type: 32 bits sized, signed integer.

```
cint64 = UnixType.cint64
```

C type: 64 bits sized, signed integer.

```
cint8 = UnixType.cint8
```

C type: 8 bits sized, signed integer.

```
clock_t = UnixType.clock_t
```

Clock ticks type.

```
clong = UnixType.clong
```

C type: long signed integer (double sized).

```
clonglong = UnixType.clonglong
```

C type: 64-bit (double long) signed integer.

```
coff_t = UnixType.TOff
```

character offset type.

```
cschar = UnixType.cschar
```

Signed character type.

```
cshort = UnixType.cshort
```

C type: short signed integer (half sized).

```
csigned = UnixType.csigned
```

`csigned` is an alias for cint (2142).

```
csint = UnixType.csint
```

Signed integer.

```
csize_t = UnixType.size_t
```

Character size type.

```
cslong = UnixType.cslong
```

The size is CPU dependent.

```
cslonglong = UnixType.cslonglong
```

`cslonglong` is an alias for clonglong (2142).

```
csshort = UnixType.csshort
```

Short signed integer type.

```
cuchar = UnixType.cuchar
```

Alias for #rtl.UnixType.cuchar (2178).

```
cuint = UnixType.cuint
```

C type: unsigned integer (natural size).

```
cuint16 = UnixType.cuint16
```

C type: 16 bits sized, unsigned integer.

```
cuint32 = UnixType.cuint32
```

C type: 32 bits sized, unsigned integer.

```
cuint64 = UnixType.cuint64
```

C type: 64 bits sized, unsigned integer.

```
cuint8 = UnixType.cuint8
```

C type: 8 bits sized, unsigned integer.

```
culong = UnixType.culong
```

C type: long unsigned integer (double sized).

```
culonglong = UnixType.culonglong
```

C type: 64-bit (double long) unsigned integer.

```
cunsigned = UnixType.cunsigned
```

Alias for #rtl.unixtype.cunsigned (2179).

```
cushort = UnixType.cushort
```

C type: short unsigned integer (half sized).

```
dev_t = UnixType.dev_t
```

Device descriptor type.

```
gid_t = UnixType.gid_t
```

Group ID type.

```
ino_t = UnixType.ino_t
```

Inode type.

```
mode_t = UnixType.mode_t
```

Inode mode type.

```
nlink_t = UnixType.nlink_t
```

Number of links type.

```
off_t = UnixType.off_t
```

Offset type.

```
pcbool = UnixType.pcbool
```

Pointer to boolean type cbool (2141)

```
pcchar = UnixType.pcchar
```

Alias for #rtl.UnixType.pcchar (2180).

```
pcdouble = UnixType.pcdouble
```

Pointer to cdouble (171) type.

```
pcfloat = UnixType.pcfloat
```

Pointer to cfloat (171) type.

```
pcint = UnixType.pcint
```

Pointer to cInt (2142) type.

```
pcint16 = UnixType.pcint16
```

Pointer to 16-bit signed integer type.

```
pcint32 = UnixType.pcint32
```

Pointer to signed 32-bit integer type.

```
pcint64 = UnixType.pcint64
```

Pointer to signed 64-bit integer type.

```
pcint8 = UnixType.pcint8
```

Pointer to 8-bits signed integer type.

```
pClock = UnixType.pClock
```

Pointer to TClock (2148) type.

```
pclong = UnixType.pclong
```

Pointer to cLong (2142) type.

```
pclonglong = UnixType.pclonglong
```

Pointer to longlong type.

```
pcschar = UnixType.pcschar
```

Pointer to character type cschar (2142).

```
pcshort = UnixType.pcshort
```

Pointer to cShort (2142) type.

```
pcsigned = UnixType.pcsigned
```

Pointer to signed integer type csigned (2142).

```
pcsint = UnixType.pcsint
```

Pointer to signed integer type csint (2143)

```
pcsize_t = UnixType.psize_t
```

Pointer to character size type `pcsize_t`.

```
pcslong = UnixType.pcslong
```

Pointer to the signed long cslong (2143)

```
pcslonglong = UnixType.pcslonglong
```

Pointer to Signed longlong type cslonglong (2143)

```
pcsshort = UnixType.pcsshort
```

Pointer to short signed integer type csshort (2143)

```
pcuchar = UnixType.pcuchar
```

Alias for #rtl.UnixType.pcuchar (2181).

```
pcuint = UnixType.pcuint
```

Pointer to cUInt (2143) type.

```
pcuint16 = UnixType.pcuint16
```

Pointer to 16-bit unsigned integer type.

```
pcuint32 = UnixType.pcuint32
```

Pointer to unsigned 32-bit integer type.

```
pcuint64 = UnixType.pcuint64
```

Pointer to unsigned 64-bit integer type.

```
pcuint8 = UnixType.pcuint8
```

Pointer to 8-bits unsigned integer type.

```
pculong = UnixType.pculong
```

Pointer to cuLong (2143) type.

```
pculonglong = UnixType.pculonglong
```

Unsigned longlong type.

```
pcunsigned = UnixType.pcunsigned
```

Alias for #rtl.unixtype.pcunsigned (2182).

```
pcushort = UnixType.pcushort
```

Pointer to cuShort (2144) type.

```
pDev = UnixType.pDev
```

Pointer to TDev (2148) type.

```
pGid = UnixType.pGid
```

Pointer to TGid (2149) type.

```
pid_t = UnixType.pid_t
```

Process ID type.

```
pIno = UnixType.pIno
```

Pointer to TIno (2149) type.

```
pMode = UnixType.pMode
```

Pointer to TMode (2149) type.

```
pnLink = UnixType.pnLink
```

Pointer to TnLink (2149) type.

```
pOff = UnixType.pOff
```

Pointer to TOff (2149) type.

```
pPid = UnixType.pPid
```

Pointer to TPid (2149) type.

```
pSize = UnixType.pSize
```

Pointer to TSize (2150) type.

```
pSize_t = UnixType.pSize_t
```

Pointer to type `Size_t`.

```
pSocklen = UnixType.pSocklen
```

Pointer to TSockLen (2150) type.

```
psSize = UnixType.psSize
```

Pointer to TsSize (2150) type.

```
pstatfs = UnixType.PStatFs
```

Pointer to `statfs` type.

```
pthread_cond_t = UnixType.pthread_cond_t
```

Thread conditional variable type.

```
pthread_mutex_t = UnixType.pthread_mutex_t
```

Thread mutex type.

```
pthread_t = UnixType.pthread_t
```

POSIX thread type.

```
pTime = UnixType.pTime
```

Pointer to TTime (2150) type.

```
ptimespec = UnixType.ptimespec
```

Pointer to timespec (2149) type.

```
ptimeval = UnixType.ptimeval
```

Pointer to timeval (2149) type.

```
ptime_t = UnixType.ptime_t
```

Pointer to time_t (2149) type.

```
pUid = UnixType.pUid
```

Pointer to TUid (2150) type.

```
size_t = UnixType.size_t
```

Size specification type.

```
socklen_t = UnixType.socklen_t
```

Socket address length type.

```
ssize_t = UnixType.ssize_t
```

Small size type.

```
TClock = UnixType.TClock
```

Alias for clock_t (2142) type.

```
TDev = UnixType.TDev
```

Alias for dev_t (2144) type.

```
TFSearchOption = (NoCurrentDirectory,CurrentDirectoryFirst,
  CurrentDirectoryLast)
```

Table 81.2: Enumeration values for type TFSearchOption

Value	Explanation
CurrentDirectoryFirst	Search the current directory first, before all directories in the search path.
CurrentDirectoryLast	Search the current directory last, after all directories in the search path.
NoCurrentDirectory	Do not search the current directory unless it is specified in the search path.

Describes the search strategy used by FSearch (2161).

```
TGid = UnixType.TGid
```

Alias for gid_t (2144) type.

```
timespec = UnixType.timespec
```

Short time specification type.

```
timeval = UnixType.timeval
```

Time specification type.

```
time_t = UnixType.time_t
```

Time span type.

```
TIno = UnixType.TIno
```

Alias for ino_t (2144) type.

```
TIOCtlRequest = UnixType.TIOCtlRequest
```

Alias for the TIOCtlRequest (2185) type in unixtypes.

```
TMode = UnixType.TMode
```

Alias for mode_t (2144) type.

```
TnLink = UnixType.TnLink
```

Alias for nlink_t (2144) type.

```
TOff = UnixType.TOff
```

Alias for off_t (2144) type.

```
TPid = UnixType.TPid
```

Alias for pid_t (2147) type.

```
TSize = UnixType.TSize
```

Alias for size_t (2148) type.

```
TSocklen = UnixType.TSocklen
```

Alias for socklen_t (2148) type.

```
TsSize = UnixType.TsSize
```

Alias for ssize_t (2148) type.

```
tstatfs = UnixType.TStatFs
```

`StatFS` returns in `Info` information about the file system on which the file `Path` resides. `Info` is of type TStatFS (2189).

The function returns zero if the call was successful, a nonzero value is returned if the call failed.

```
TTime = UnixType.TTime
```

Alias for TTime (2150) type.

```
Ttimespec = UnixType.Ttimespec
```

Alias for TimeSpec (2149) type.

```
TTimeVal = UnixType.TTimeVal
```

Alias for timeval (2149) type.

```
TUid = UnixType.TUid
```

Alias for uid_t (2150) type.

```
uid_t = UnixType.uid_t
```

User ID type.

81.3 Procedures and functions

81.3.1 AssignPipe

Synopsis: Create a set of pipe file handlers.

Declaration:
```
function AssignPipe(var pipe_in: cint; var pipe_out: cint) : cint
function AssignPipe(var pipe_in: text; var pipe_out: text) : cint
function AssignPipe(var pipe_in: File; var pipe_out: File) : cint
```

Visibility: default

Description: `AssignePipe` creates a pipe, i.e. two file objects, one for input, one for output. What is written to `Pipe_out`, can be read from `Pipe_in`.

This call is overloaded. The in and out pipe can take three forms: an typed or untyped file, a text file or a file descriptor.

If a text file is passed then reading and writing from/to the pipe can be done through the usual `Readln(Pipe_in,...)` and `Writeln(Pipe_out,...)` procedures.

The function returns `True` if everything went successfully, `False` otherwise.

Errors: In case the function fails and returns `False`, extended error information is returned by the FpGetErrno (198) function:

sys_emfile Too many file descriptors for this process.

sys_enfile The system file table is full.

See also: POpen (2165), #rtl.baseunix.FpMkFifo (207)

Listing: ./examples/unixex/ex36.pp

```
Program Example36;

{ Program to demonstrate the AssignPipe function. }

Uses BaseUnix,Unix;

Var pipi,pipo : Text;
    s : String;

begin
  Writeln ('Assigning Pipes.');
  If assignpipe(pipi,pipo)<>0 then
    Writeln('Error assigning pipes !',fpgeterrno);
  Writeln ('Writing to pipe, and flushing.');
  Writeln (pipo,'This is a textstring');close(pipo);
  Writeln ('Reading from pipe.');
  While not eof(pipi) do
    begin
    Readln (pipi,s);
    Writeln ('Read from pipe : ',s);
    end;
  close (pipi);
  writeln ('Closed pipes.');
  writeln
end.
```

81.3.2 AssignStream

Synopsis: Assign stream for in and output to a program.

Declaration:
```
function AssignStream(var StreamIn: text; var Streamout: text;
                      const Prog: ansiString;
                      const args: Array of ansistring) : cint
function AssignStream(var StreamIn: text; var Streamout: text;
                      var streamerr: text; const Prog: ansiString;
                      const args: Array of ansistring) : cint
```

Visibility: default

Description: `AssignStream` creates a 2 or 3 pipes, i.e. two (or three) file objects, one for input, one for output,(and one for standard error) the other ends of these pipes are connected to standard input and output (and standard error) of `Prog`. `Prog` is the path of a program (including path). The options for the program can be specified in `Args`.

What is written to `StreamOut`, will go to the standard input of `Prog`. Whatever is written by `Prog` to it's standard output can be read from `StreamIn`. Whatever is written by `Prog` to it's standard error read from `StreamErr`, if present.

Reading and writing happens through the usual `Readln(StreamIn,...)` and `Writeln (StreamOut,...)` procedures.

Remark You should *not* use `Reset` or `Rewrite` on a file opened with `POpen`. This will close the file before re-opening it again, thereby closing the connection with the program.

The function returns the process ID of the spawned process, or -1 in case of error.

Errors: Extended error information is returned by the FpGetErrno (198) function.

sys_emfile Too many file descriptors for this process.

sys_enfile The system file table is full.

Other errors include the ones by the fork and exec programs

See also: AssignPipe (2150), POpen (2165)

Listing: ./examples/unixex/ex38.pp

```
Program Example38;

{ Program to demonstrate the AssignStream function. }

Uses BaseUnix,Unix;

Var Si,So : Text;
    S : String;
    i : longint;

begin
  if not (paramstr(1)='-son') then
    begin
    Writeln ('Calling son');
    Assignstream (Si,So,paramstr(0),['-son']);
    if fpgeterrno<>0 then
      begin
      writeln ('AssignStream failed !');
      halt(1);
      end;
    Writeln ('Speaking to son');
    For i:=1 to 10 do
      begin
      writeln (so,'Hello son !');
      if ioresult<>0 then writeln ('Can''t speak to son...');
      end;
    For i:=1 to 3 do writeln (so,'Hello chap !');
    close (so);
    while not eof(si) do
      begin
```

```
      readln (si,s);
      writeln ('Father: Son said : ',S);
      end;
    WriteIn ('Stopped conversation');
    Close (Si);
    WriteIn ('Put down phone');
    end
  Else
    begin
    WriteIn ('This is the son ');
    While not eof (input) do
      begin
      readln (s);
      if pos ('Hello son !',S)<>0 then
         WriteIn ('Hello Dad !')
      else
         writeln ('Who are you ?');
      end;
    close (output);
    end
end.
```

81.3.3 EpochToLocal

Declaration: `procedure EpochToLocal(epoch: Int64; var year: Word; var month: Word; var day: Word; var hour: Word; var minute: Word; var second: Word)`

Visibility: default

81.3.4 EpochToUniversal

Declaration: `procedure EpochToUniversal(epoch: Int64; var year: Word; var month: Word; var day: Word; var hour: Word; var minute: Word; var second: Word)`

Visibility: default

81.3.5 FpExecL

Synopsis: Execute process (using argument list, environment).

Declaration: `function FpExecL(const PathName: RawByteString; const S: Array of RawByteString) : cint`

Visibility: default

Description: `FpExecL` replaces the currently running program with the program, specified in `PathName`. `S` is an array of command options. The executable in `PathName` must be an absolute pathname. The current process' environment is passed to the program. On success, `FpExecL` does not return.

Errors: Extended error information is returned by the FpGetErrno (198) function:

sys_eacces File is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_eperm The file system is mounted *noexec*.

sys_e2big Argument list too big.

sys_enoexec The magic number in the file is incorrect.

sys_enoent The file does not exist.

sys_enomem Not enough memory for kernel, or to split command line.

sys_enotdirA component of the path is not a directory.

sys_eloop The path contains a circular reference (via symlinks).

See also: FpExecve (192), FpExecv (2156), FpExecvp (2157), FpExecle (2154), FpExeclp (2155), FpFork (195)

Listing: ./examples/unixex/ex77.pp

```
Program Example77;

{ Program to demonstrate the FPExecL function. }

Uses Unix, strings;

begin
  { Execute 'ls -l', with current environment. }
  { 'ls' is NOT looked for in PATH environment variable.}
  FpExecL ('/bin/ls',['-l']);
end.
```

81.3.6 FpExecLE

Synopsis: Execute process (using argument list, environment).

Declaration:
```
function FpExecLE(const PathName: RawByteString;
                  const S: Array of RawByteString; MyEnv: PPChar) : cint
```

Visibility: default

Description: `FpExecLE` replaces the currently running program with the program, specified in `PathName`. `S` is an array of command options. The executable in `PathName` must be an absolute pathname. The environment in `MyEnv` is passed to the program. On success, `FpExecLE` does not return.

Errors: Extended error information is returned by the FpGetErrno (198) function:

sys_eaccess File is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_eperm The file system is mounted *noexec*.

sys_e2big Argument list too big.

sys_enoexec The magic number in the file is incorrect.

sys_enoent The file does not exist.

sys_enomem Not enough memory for kernel, or to split command line.

sys_enotdirA component of the path is not a directory.

sys_eloop The path contains a circular reference (via symlinks).

See also: FpExecve (192), FpExecv (2156), FpExecvp (2157), FpExecl (2153), FpExeclp (2155), FpFork (195)

Listing: ./examples/unixex/ex11.pp

```
Program Example11;

{ Program to demonstrate the Execle function. }

Uses Unix, strings;

begin
  { Execute 'ls -l', with current environment. }
  { 'ls' is NOT looked for in PATH environment variable.}
  { envp is defined in the system unit.}
  Execle ('/bin/ls -l',envp);
end.
```

81.3.7 FpExecLP

Synopsis: Execute process (using argument list, environment; search path).

Declaration:
```
function FpExecLP(const PathName: RawByteString;
                  const S: Array of RawByteString) : cint
```

Visibility: default

Description: `FpExecLP` replaces the currently running program with the program, specified in `PathName`. `S` is an array of command options. The executable in `PathName` is searched in the path, if it isn't an absolute filename. The current environment is passed to the program. On success, `FpExecLP` does not return.

Errors: Extended error information is returned by the FpGetErrno (198) function:

sys_eaccesFile is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_epermThe file system is mounted *noexec*.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel, or to split command line.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: FpExecve (192), FpExecv (2156), FpExecvp (2157), FpExecle (2154), FpExecl (2153), FpFork (195)

Listing: ./examples/unixex/ex76.pp

```
Program Example76;

{ Program to demonstrate the FpExeclp function. }

Uses Unix, strings;

begin
  { Execute 'ls -l', with current environment. }
```

```
  { 'ls' is looked for in PATH environment variable.}
  { envp is defined in the system unit.}
  FpExeclp ('ls',['-l']);
end.
```

81.3.8 FpExecLPE

Synopsis: Execute a program in the path, and pass it an environment.

Declaration:
```
function FpExecLPE(const PathName: RawByteString;
                   const S: Array of RawByteString; env: PPChar) : cint
```

Visibility: default

Description: `FpExecLPE` does the same as FpExecLP (2155), but additionally it specifies the environment for the new process in `env`, a pointer to a null-terminated array of null-terminated strings.

Errors: On success, this function does not return.

See also: FpExecLP (2155), FpExecLE (2154)

81.3.9 FpExecV

Synopsis: Execute process.

Declaration:
```
function FpExecV(const PathName: RawByteString; args: PPChar) : cint
```

Visibility: default

Description: `FpExecV` replaces the currently running program with the program, specified in `PathName`. It gives the program the options in `args`. This is a pointer to an array of pointers to null-terminated strings. The last pointer in this array should be nil. The current environment is passed to the program. On success, `FpExecV` does not return.

Errors: Extended error information is returned by the FpGetErrno (198) function:

sys_eacces File is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_eperm The file system is mounted *noexec*.

sys_e2big Argument list too big.

sys_enoexec The magic number in the file is incorrect.

sys_enoent The file does not exist.

sys_enomem Not enough memory for kernel.

sys_enotdirA component of the path is not a directory.

sys_eloop The path contains a circular reference (via symlinks).

See also: FpExecve (192), FpExecvp (2157), FpExecle (2154), FpExecl (2153), FpExeclp (2155), FpFork (195)

Listing: ./examples/unixex/ex8.pp

```
Program Example8;

{ Program to demonstrate the Execv function. }

Uses Unix, strings;

Const Arg0 : PChar = '/bin/ls';
      Arg1 : Pchar = '-l';

Var PP : PPchar;


begin
  GetMem (PP,3*SizeOf(Pchar));
  PP[0]:=Arg0;
  PP[1]:=Arg1;
  PP[3]:=Nil;
  { Execute '/bin/ls -l', with current environment }
  fpExecv ('/bin/ls',pp);
end.
```

81.3.10 FpExecVP

Synopsis: Execute process, search path.

Declaration: `function FpExecVP(const PathName: RawByteString; args: PPChar) : cint`

Visibility: default

Description: `FpExecVP` replaces the currently running program with the program, specified in `PathName`. The executable in `path` is searched in the path, if it isn't an absolute filename. It gives the program the options in `args`. This is a pointer to an array of pointers to null-terminated strings. The last pointer in this array should be nil. The current environment is passed to the program. On success, `execvp` does not return.

Errors: Extended error information is returned by the FpGetErrno (198) function:

sys_eacces File is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_eperm The file system is mounted *noexec*.

sys_e2big Argument list too big.

sys_enoexec The magic number in the file is incorrect.

sys_enoent The file does not exist.

sys_enomem Not enough memory for kernel.

sys_enotdirA component of the path is not a directory.

sys_eloop The path contains a circular reference (via symlinks).

See also: FpExecve (192), FpExecv (2156), FpExecle (2154), FpExecl (2153), FpExeclp (2155), FpFork (195)

Listing: ./examples/unixex/ex79.pp

```
Program Example79;

{ Program to demonstrate the FpExecVP function. }

Uses Unix, strings;

Const Arg0 : PChar = 'ls';
      Arg1 : Pchar = '-l';

Var PP : PPchar;


begin
  GetMem (PP,3*SizeOf(Pchar));
  PP[0]:=Arg0;
  PP[1]:=Arg1;
  PP[2]:=Nil;
  { Execute 'ls -l', with current environment. }
  { 'ls' is looked for in PATH environment variable.}
  fpExecvp ('ls',pp);
end.
```

81.3.11 FpExecVPE

Synopsis: Execute process, search path using environment.

Declaration:
```
function FpExecVPE(const PathName: RawByteString; args: PPChar;
                   env: PPChar) : cint
```

Visibility: default

Description: `FpExecVP` replaces the currently running program with the program, specified in `PathName`. The executable in `path` is searched in the path, if it isn't an absolute filename. It gives the program the options in `args`. This is a pointer to an array of pointers to null-terminated strings. The last pointer in this array should be nil. The environment in `Env` is passed to the program. On success, `execvp` does not return.

Errors: Extended error information is returned by the FpGetErrno (198) function:

sys_eacces File is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_eperm The file system is mounted *noexec*.

sys_e2big Argument list too big.

sys_enoexec The magic number in the file is incorrect.

sys_enoent The file does not exist.

sys_enomem Not enough memory for kernel.

sys_enotdirA component of the path is not a directory.

sys_eloop The path contains a circular reference (via symlinks).

See also: FpExecve (192), FpExecv (2156), FpExecle (2154), FpExecl (2153), FpExeclp (2155), FpFork (195)

Listing: ./examples/unixex/ex79.pp

```
Program Example79;

{ Program to demonstrate the FpExecVP function. }

Uses Unix, strings;

Const Arg0 : PChar = 'ls';
      Arg1 : Pchar = '-l';

Var PP : PPchar;


begin
  GetMem (PP,3*SizeOf(Pchar));
  PP[0]:=Arg0;
  PP[1]:=Arg1;
  PP[2]:=Nil;
  { Execute 'ls -l', with current environment. }
  { 'ls' is looked for in PATH environment variable.}
  fpExecvp ('ls',pp);
end.
```

81.3.12 fpFlock

Synopsis: Lock a file (advisory lock).

Declaration:
```
function fpFlock(var T: text; mode: cint) : cint
function fpFlock(var F: File; mode: cint) : cint
function fpFlock(fd: cint; mode: cint) : cint
```

Visibility: default

Description: `FpFLock` implements file locking. it sets or removes a lock on the file `F`. F can be of type `Text` or `File`, or it can be a Linux file descriptor (a longint) `Mode` can be one of the following constants :

LOCK_SHsets a shared lock.

LOCK_EXsets an exclusive lock.

LOCK_UNunlocks the file.

LOCK_NB This can be OR-ed together with the other. If this is done the application doesn't block when locking.

The function returns zero if successful, a nonzero return value indicates an error.

Errors: Extended error information is returned by the FpGetErrno (198) function:

See also: #rtl.baseunix.FpFcntl (193), fpFSync (2159)

81.3.13 fpfStatFS

Synopsis: Retrieve file system information.

Declaration:
```
function fpfStatFS(Fd: cint; Info: pstatfs) : cint
```

Visibility: default

Description: `fpStatFS` returns in `Info` information about the file system on which the open file descriptor `fd` resides. Info is of type `tstatfs`. The function returns `0` if the call was successful, or an error code if the call failed.

Errors: On error, a non-zero error code is returned

See also: fpStatFS (2160), #rtl.baseunix.fpfStat (195)

81.3.14 fpfsync

Synopsis: Synchronize file's kernel data with disk.

Declaration:
```
function fpfsync(fd: cint) : cint
```

Visibility: default

Description: `fpFSync` synchronizes the kernel data for file `fd` (the cache) with the disk. The call will not return till all file data was written to disk.

If the call was successful, 0 is returned. On failure, a nonzero value is returned.

Errors: Extended error information is returned by the FpGetErrno (198) function:

See also: FpFLock (2159)

81.3.15 fpgettimeofday

Synopsis: Return kernel time of day in GMT.

Declaration:
```
function fpgettimeofday(tp: ptimeval; tzp: ptimezone) : cint
```

Visibility: default

Description: `FpGetTimeOfDay` returns the number of seconds since 00:00, January 1 1970, GMT in a `timeval` record. This time NOT corrected any way, not taking into account timezones, daylight savings time and so on.

It is simply a wrapper to the kernel system call.

Errors: None.

81.3.16 fpStatFS

Synopsis: Retrieve file system information.

Declaration:
```
function fpStatFS(Path: PChar; Info: pstatfs) : cint
function fpStatFS(Path: ansistring; Info: pstatfs) : cint
```

Visibility: default

Description: `fpStatFS` returns in `Info` information about the file system on which the file or path `Path` resides. Info is of type `tstatfs`. The function returns `0` if the call was successful, or an error code if the call failed.

Errors: On error, a non-zero error code is returned

See also: fpFStatFS (2159), #rtl.baseunix.fpStat (228)

81.3.17 fpSystem

Synopsis: Execute and feed command to system shell.

Declaration:
```
function fpSystem(const Command: RawByteString) : cint
```

Visibility: default

Description: `FpSystem` invokes the bash shell (/bin/sh), and feeds it the command `Command` (using the `-c` option). The function then waits for the command to complete, and then returns the exit status of the command in wait(3) format, or 127 if it could not complete the FpFork (195) or FpExecve (192) calls. To convert the return value of fpsystem to the real return value use WEXITSTATUS() (237).

Errors: Errors are reported in (fpget)ErrNo (198)

See also: POpen (2165), FpFork (195), FpExecve (192)

Listing: ./examples/unixex/ex80.pp

```
program example56;

uses Unix;

{ Program to demonstrate the Shell function }

Var S : Longint;

begin
  WriteIn ('Output of ls -l *.pp');
  S:=fpSystem('ls -l *.pp');
  WriteIn ('Command exited with status : ',S);
end.
```

81.3.18 FSearch

Synopsis: Search for file in search path.

Declaration:
```
function FSearch(const path: RawByteString; dirlist: RawByteString;
                 CurrentDirStrategy: TFSearchOption) : RawByteString
function FSearch(const path: RawByteString; dirlist: RawByteString)
                  : RawByteString
function FSearch(const path: UnicodeString; dirlist: UnicodeString;
                 CurrentDirStrategy: TFSearchOption) : UnicodeString
function FSearch(const path: UnicodeString; dirlist: UnicodeString)
                  : UnicodeString
```

Visibility: default

Description: `FSearch` searches in `DirList`, a colon separated list of directories, for a file named `Path`. It then returns a path to the found file.

The `CurrentDirStrategy` determines how the current directory is treated when searching:

NoCurrentDirectoryDo not search the current directory unless it is specified in the search path.

CurrentDirectoryFirstSearch the current directory first, before all directories in the search path.

CurrentDirectoryLastSearch the current directory last, after all directories in the search path.

It is mainly provided to mimic DOS search path behaviour. Default behaviour is to search the current directory first.

Errors: An empty string if no such file was found.

See also: #rtl.unixutil.FNMatch (2191)

Listing: ./examples/unixex/ex46.pp

```
Program Example46;

{ Program to demonstrate the FSearch function. }

Uses BaseUnix, Unix, Strings;

begin
  WriteLn ('ls is in : ',FSearch ('ls',strpas(fpGetenv('PATH'))));
end.
```

81.3.19 GetDomainName

Synopsis: Return current domain name.

Declaration: `function GetDomainName : string`

Visibility: default

Description: Get the domain name of the machine on which the process is running. An empty string is returned if the domain is not set.

Errors: None.

See also: GetHostName (2162)

Listing: ./examples/unixex/ex39.pp

```
Program Example39;

{ Program to demonstrate the GetDomainName function. }

Uses Unix;

begin
  WriteLn ('Domain name of this machine is : ',GetDomainName);
end.
```

81.3.20 GetHostName

Synopsis: Return host name.

Declaration: `function GetHostName : string`

Visibility: default

Description: Get the hostname of the machine on which the process is running. An empty string is returned if hostname is not set.

Errors: None.

See also: GetDomainName (2162)

Listing: ./examples/unixex/ex40.pp

```
Program Example40;

{ Program to demonstrate the GetHostName function. }

Uses unix;

begin
  WriteLn ('Name of this machine is : ',GetHostName);
end.
```

81.3.21 GetLocalTimezone

Synopsis: Return local timezone information.

Declaration:
```
function GetLocalTimezone(timer: Int64; timerIsUTC: Boolean;
                          var ATZInfo: TTZInfo; var ATZInfoEx: TTZInfoEx)
                           : Boolean
function GetLocalTimezone(timer: Int64; timerIsUTC: Boolean;
                          var ATZInfo: TTZInfo) : Boolean
```

Visibility: default

Description: `GetLocalTimeZone` returns the local timezone information. It also initializes the `TZSeconds` variable, which is used to correct the epoch time to local time.

There should never be any need to call this function directly. It is called by the initialization routines of the Linux unit.

See also: GetTimezoneFile (2163), ReadTimezoneFile (2166)

81.3.22 GetTimezoneFile

Synopsis: Return name of timezone information file.

Declaration:
```
function GetTimezoneFile : string
```

Visibility: default

Description: `GetTimezoneFile` returns the location of the current timezone file. The location of file is determined as follows:

1. If /etc/timezone exists, it is read, and the contents of this file is returned. This should work on Debian systems.
2. If /usr/lib/zoneinfo/localtime exists, then it is returned. (this file is a symlink to the timezone file on SuSE systems)
3. If /etc/localtime exists, then it is returned. (this file is a symlink to the timezone file on RedHat systems)

Errors: If no file was found, an empty string is returned.

See also: ReadTimezoneFile (2166)

81.3.23 Gettzdaylight

Declaration: `function Gettzdaylight : Boolean`

Visibility: default

81.3.24 GetTZInfo

Declaration: `function GetTZInfo : TTZInfo`

Visibility: default

81.3.25 GetTZInfoEx

Declaration: `function GetTZInfoEx : TTZInfoEx`

Visibility: default

81.3.26 Gettzname

Declaration: `function Gettzname(const b: Boolean) : string`

Visibility: default

81.3.27 GetTzseconds

Declaration: `function GetTzseconds : LongInt`

Visibility: default

81.3.28 GregorianToJulian

Declaration:
```
function GregorianToJulian(Year: LongInt; Month: LongInt; Day: LongInt)
                                : LongInt
```

Visibility: default

81.3.29 JulianToGregorian

Declaration:
```
procedure JulianToGregorian(JulianDN: LongInt; var Year: Word;
                                var Month: Word; var Day: Word)
```

Visibility: default

81.3.30 LocalToEpoch

Declaration:
```
function LocalToEpoch(year: Word; month: Word; day: Word; hour: Word;
                          minute: Word; second: Word) : Int64
```

Visibility: default

81.3.31 PClose

Synopsis: Close file opened with POpen (2165).

Declaration:
```
function PClose(var F: File) : cint
function PClose(var F: text) : cint
```

Visibility: default

Description: `PClose` closes a file opened with POpen (2165). It waits for the command to complete, and then returns the exit status of the command.

For an example, see POpen (2165)

Errors: Extended error information is returned by the FpGetErrno (198) function.

See also: POpen (2165)

81.3.32 POpen

Synopsis: Pipe file to standard input/output of program.

Declaration:
```
function POpen(var F: text; const Prog: RawByteString; rw: Char) : cint
function POpen(var F: File; const Prog: RawByteString; rw: Char) : cint
function POpen(var F: text; const Prog: UnicodeString; rw: Char) : cint
function POpen(var F: File; const Prog: UnicodeString; rw: Char) : cint
```

Visibility: default

Description: `POpen` runs the command specified in `Prog`, and redirects the standard in or output of the command to the other end of the pipe `F`. The parameter `rw` indicates the direction of the pipe. If it is set to `'W'`, then F can be used to write data, which will then be read by the command from stdinput. If it is set to `'R'`, then the standard output of the command can be read from `F`. `F` should be reset or rewritten prior to using it. `F` can be of type `Text` or `File`. A file opened with `POpen` can be closed with `Close`, but also with PClose (2164). The result is the same, but `PClose` returns the exit status of the command `Prog`.

Errors: Extended error information is returned by the FpGetErrno (198) function. Errors are essentially those of the Execve, Dup and AssignPipe commands.

See also: AssignPipe (2150), PClose (2164)

Listing: ./examples/unixex/ex37.pp

```
Program Example37;

{ Program to demonstrate the Popen function. }

uses BaseUnix,Unix;

var f : text;
    i : longint;

begin
  writeln ('Creating a shell script to which echoes its arguments');
  writeln ('and input back to stdout');
  assign (f,'test21a');
  rewrite (f);
  writeln (f,'#!/bin/sh');
```

```
  writeln (f,'echo this is the child speaking.... ');
  writeln (f,'echo got arguments \*"$*"\* ');
  writeln (f,'cat');
  writeln (f,'exit 2');
  writeln (f);
  close (f);
  fpchmod ('test21a',&755);
  popen (f,'./test21a arg1 arg2','W');
  if fpgeterrno<>0 then
     writeln ('error from POpen : errno : ', fpgeterrno);
  for i:=1 to 10 do
    writeln (f,'This is written to the pipe, and should appear on stdout.');
  Flush(f);
  Writeln ('The script exited with status : ',PClose (f));
  writeln;
  writeln ('Press <return> to remove shell script.');
  readln;
  assign (f,'test21a');
  erase (f)
end.
```

81.3.33 ReadTimezoneFile

Synopsis: Read the timezone file and initialize time routines.

Declaration: `function ReadTimezoneFile(fn: string) : Boolean`

Visibility: default

Description: `ReadTimeZoneFile` reads the timezone file `fn` and initializes the local time routines based on the information found there.

There should be no need to call this function. The initialization routines of the linux unit call this routine at unit startup.

Errors: None.

See also: GetTimezoneFile (2163), GetLocalTimezone (2163)

81.3.34 RefreshTZInfo

Declaration: `procedure RefreshTZInfo`

Visibility: default

81.3.35 ReReadLocalTime

Synopsis: Re-Read the local time files.

Declaration: `procedure ReReadLocalTime`

Visibility: default

Description: `ReReadLocalTime` can be used to re-initialize the local timezone information.

To speed up conversion of epoch (UTC) time to local time, the timezone information is loaded only once, at program startup. Calling this routine re-reads the timezone information using current timezone settings.

The EpochToLocal (2191) function uses timezone information to transform epoch time to local time. This timezone information does not change while the application is running: in particular, on DST transitions or when the timezone files change, the time returned by local time routines will be wrong.

See also: Date (1694), Time (1801), Now (1765), EpochToLocal (2191)

81.3.36 SeekDir

Synopsis: Seek to position in directory.

Declaration: `procedure SeekDir(p: pDir; loc: clong)`

Visibility: default

Description: `SeekDir` sets the directory pointer to the `loc`-th entry in the directory structure pointed to by `p`.

For an example, see #rtl.baseunix.fpOpenDir (212).

Errors: Extended error information is returned by the FpGetErrno (198) function:

See also: #rtl.baseunix.fpCloseDir (189), #rtl.baseunix.fpReadDir (216), #rtl.baseunix.fpOpenDir (212), TellDir (2167)

81.3.37 SetTZInfo

Declaration: `procedure SetTZInfo(const ATZInfo: TTZInfo; const ATZInfoEx: TTZInfoEx)`

Visibility: default

81.3.38 TellDir

Synopsis: Return current location in a directory.

Declaration: `function TellDir(p: pDir) : TOff`

Visibility: default

Description: `TellDir` returns the current location in the directory structure pointed to by `p`. It returns -1 on failure.

For an example, see #rtl.baseunix.fpOpenDir (212).

See also: #rtl.baseunix.fpCloseDir (189), #rtl.baseunix.fpReadDir (216), #rtl.baseunix.fpOpenDir (212), SeekDir (2166)

81.3.39 UniversalToEpoch

Declaration:
```
function UniversalToEpoch(year: Word; month: Word; day: Word;
                          hour: Word; minute: Word; second: Word) : Int64
```

Visibility: default

81.3.40 WaitProcess

Synopsis: Wait for process to terminate.

Declaration: `function WaitProcess(Pid: cint) : cint`

Visibility: default

Description: `WaitProcess` waits for process `PID` to exit. `WaitProcess` is equivalent to the #rtl.baseunix.FpWaitPID (236) call:

```
FpWaitPid(PID,@result,0)
```

Handles of Signal interrupts (errno=EINTR), and returns the Exitcode of Process `PID` (>=0) or -Status if it was terminated

Errors: None.

See also: #rtl.baseunix.FpWaitPID (236), #rtl.baseunix.WTERMSIG (238), #rtl.baseunix.WSTOPSIG (238), #rtl.baseunix.WIFEXITED (237), WIFSTOPPED (2168), #rtl.baseunix.WIFSIGNALED (238), W_EXITCODE (2168), W_STOPCODE (2168), #rtl.baseunix.WEXITSTATUS (237)

81.3.41 WIFSTOPPED

Synopsis: Check whether the process is currently stopped.

Declaration: `function WIFSTOPPED(Status: Integer) : Boolean`

Visibility: default

Description: `WIFSTOPPED` checks `Status` and returns `true` if the process is currently stopped. This is only possible if WUNTRACED was specified in the options of FpWaitPID (236).

See also: #rtl.baseunix.FpWaitPID (236), WaitProcess (2167), #rtl.baseunix.WTERMSIG (238), #rtl.baseunix.WSTOPSIG (238), #rtl.baseunix.WIFEXITED (237), #rtl.baseunix.WIFSIGNALED (238), W_EXITCODE (2168), W_STOPCODE (2168), #rtl.baseunix.WEXITSTATUS (237)

81.3.42 W_EXITCODE

Synopsis: Construct an exit status based on an return code and signal.

Declaration: `function W_EXITCODE(ReturnCode: Integer; Signal: Integer) : Integer`

Visibility: default

Description: `W_EXITCODE` combines `ReturnCode` and `Signal` to a status code fit for `WaitPid`.

See also: #rtl.baseunix.FpWaitPID (236), WaitProcess (2167), #rtl.baseunix.WTERMSIG (238), #rtl.baseunix.WSTOPSIG (238), #rtl.baseunix.WIFEXITED (237), WIFSTOPPED (2168), #rtl.baseunix.WIFSIGNALED (238), W_EXITCODE (2168), W_STOPCODE (2168), #rtl.baseunix.WEXITSTATUS (237)

81.3.43 W_STOPCODE

Synopsis: Construct an exit status based on a signal.

Declaration: `function W_STOPCODE(Signal: Integer) : Integer`

Visibility: default

Description: `W_STOPCODE` constructs an exit status based on `Signal`, which will cause WIFSIGNALED (238) to return `True`

See also: #rtl.baseunix.FpWaitPID (236), WaitProcess (2167), #rtl.baseunix.WTERMSIG (238), #rtl.baseunix.WSTOPSIG (238), #rtl.baseunix.WIFEXITED (237), WIFSTOPPED (2168), #rtl.baseunix.WIFSIGNALED (238), W_EXITCODE (2168), #rtl.baseunix.WEXITSTATUS (237)

81.4 TTZInfo

```
TTZInfo = record
  daylight : Boolean;
  seconds : LongInt;
  validsince
  : Int64;
  validuntil : Int64;
end
```

81.5 TTZInfoEx

```
TTZInfoEx = record
  name : Array[boolean] of RawByteString;
  leap_correct
  : LongInt;
  leap_hit : LongInt;
end
```

Chapter 82

Reference for unit 'unixcp'

82.1 Used units

Table 82.1: Used units by unit 'unixcp'

Name	Page
BaseUnix	146
System	1362

82.2 Overview

The unixcp unit provides routines to handle mapping of code page names to numerical values as used in libiconv>. The GetCodepageByName (2173) function is the main function for this. The GetCodepageData (2174) can be used to map a code page number to a name. These function can be used for instance to map code page information in environment variables to code page numbers used in string encodings. The supported code page names are the ones commonly in use in libiconv.

This unit is used for example in unit cwstring (596).

82.3 Constants, types and variables

82.3.1 Constants

```
UnixCpMap : Array[-1..UnixCpMapLimit] of TUnixCpData = ((cp: 0; name
  : 'UTF-8'), (cp: 37; name: 'IBM037'), (cp: 37; name: 'IBM-037'),
  (cp: 154; name: 'CP154'), (cp: 154; name: 'CYRILLIC-ASIAN'), (cp:
  154; name: 'PT154'), (cp: 154; name: 'PTCP154'), (cp: 154; name:
  'CSPTCP154'), (cp: 437; name: '437'), (cp: 437; name: 'CP437'),
  (cp: 437; name: 'IBM-437'), (cp: 437; name: 'CSPC8CODEPAGE437'),
  (cp: 437; name: 'IBM437'), (cp: 500; name: 'IBM500'), (cp: 500; name
  : 'IBM-500'), (cp: 708; name: 'ASMO-708'), (cp: 720; name: 'DOS-720'
  ), (cp: 737; name: 'CP737'), (cp: 737; name: 'ibm737'), (cp: 775;
  name: 'CP775'), (cp: 775; name: 'IBM775'), (cp: 775; name: 'CSPC775BALTIC'
  ), (cp: 775; name: 'ibm775'), (cp: 850; name: '850'), (cp: 850; name
  : 'CP850'), (cp: 850; name: 'IBM850'), (cp: 850; name: 'CSPC850MULTILINGUAL'
```

```
), (cp: 850; name: 'ibm850'), (cp: 852; name: '852'), (cp: 852; name
: 'CP852'), (cp: 852; name: 'IBM852'), (cp: 852; name: 'CSPCP852'
), (cp: 852; name: 'ibm852'), (cp: 853; name: 'CP853'), (cp: 855;
name: '855'), (cp: 855; name: 'CP855'), (cp: 855; name: 'IBM855'
), (cp: 855; name: 'CSIBM855'), (cp: 855; name: 'IBM855'), (cp: 857
; name: '857'), (cp: 857; name: 'CP857'), (cp: 857; name: 'IBM857'
), (cp: 857; name: 'CSIBM857'), (cp: 857; name: 'ibm857'), (cp: 858
; name: 'CP858'), (cp: 858; name: 'IBM00858'), (cp: 860; name: '860'
), (cp: 860; name: 'CP860'), (cp: 860; name: 'IBM860'), (cp: 860;
name: 'CSIBM860'), (cp: 860; name: 'IBM860'), (cp: 861; name: '861'
), (cp: 861; name: 'CP-IS'), (cp: 861; name: 'CP861'), (cp: 861; name
: 'IBM861'), (cp: 861; name: 'CSIBM861'), (cp: 861; name: 'ibm861'
), (cp: 862; name: '862'), (cp: 862; name: 'CP862'), (cp: 862; name
: 'IBM862'), (cp: 862; name: 'CSPC862LATINHEBREW'), (cp: 862; name
: 'DOS-862'), (cp: 863; name: '863'), (cp: 863; name: 'CP863'), (cp
: 863; name: 'CSIBM863'), (cp: 863; name: 'IBM863'), (cp: 864; name
: 'CP864'), (cp: 864; name: 'CSIBM864'), (cp: 864; name: 'IBM864'
), (cp: 865; name: '865'), (cp: 865; name: 'IBM-865'), (cp: 865; name
: 'CP865'), (cp: 865; name: 'CSIBM865'), (cp: 865; name: 'IBM865'
), (cp: 866; name: '866'), (cp: 866; name: 'CP866'), (cp: 866; name
: 'IBM866'), (cp: 866; name: 'CSIBM866'), (cp: 866; name: 'cp866'
), (cp: 869; name: '869'), (cp: 869; name: 'IBM-869'), (cp: 869; name
: 'CP-GR'), (cp: 869; name: 'CP869'), (cp: 869; name: 'IBM869'),
(cp: 869; name: 'CSIBM869'), (cp: 869; name: 'ibm869'), (cp: 870;
name: 'IBM870'), (cp: 874; name: 'CP874'), (cp: 874; name: 'WINDOWS-874'
), (cp: 874; name: 'windows-874'), (cp: 875; name: 'cp875'), (cp:
932; name: 'CP932'), (cp: 932; name: 'IBM-943'), (cp: 932; name:
'MS932'), (cp: 932; name: 'SHIFFT_JIS'), (cp: 932; name: 'SHIFFT_JIS-MS'
), (cp: 932; name: 'SJIS'), (cp: 932; name: 'SJIS-MS'), (cp: 932;
name: 'SJIS-OPEN'), (cp: 932; name: 'SJIS-WIN'), (cp: 932; name:
'WINDOWS-31J'), (cp: 932; name: 'WINDOWS-932'), (cp: 932; name: 'CSWINDOWS31J'
), (cp: 932; name: 'shift_jis'), (cp: 932; name: 'shift-jis'), (cp
: 936; name: 'CP936'), (cp: 936; name: 'GBK'), (cp: 936; name: 'MS936'
), (cp: 936; name: 'WINDOWS-936'), (cp: 936; name: 'gb2312'), (cp
: 949; name: 'CP949'), (cp: 949; name: 'UHC'), (cp: 949; name: 'EUC-KR'
), (cp: 949; name: 'ks_c_5601-1987'), (cp: 950; name: 'CP950'), (cp
: 950; name: 'BIG5'), (cp: 950; name: 'big5'), (cp: 1026; name: 'IBM1026'
), (cp: 1047; name: 'IBM01047'), (cp: 1125; name: 'CP1125'), (cp:
1125; name: 'IBM-1125'), (cp: 1133; name: 'CP1133'), (cp: 1133; name
: 'IBM-1133'), (cp: 1133; name: 'IBM-CP1133'), (cp: 1140; name: 'IBM01140'
), (cp: 1141; name: 'IBM01141'), (cp: 1142; name: 'IBM01142'), (cp
: 1143; name: 'IBM01143'), (cp: 1144; name: 'IBM01144'), (cp: 1145
; name: 'IBM01145'), (cp: 1146; name: 'IBM01146'), (cp: 1147; name
: 'IBM01147'), (cp: 1148; name: 'IBM01148'), (cp: 1149; name: 'IBM01149'
), (cp: 1200; name: 'UTF-16LE'), (cp: 1200; name: 'UTF16LE'), (cp
: 1200; name: 'UCS-2LE'), (cp: 1200; name: 'CP1200'), (cp: 1200; name
: 'UTF16'), (cp: 1200; name: 'UTF-16'), (cp: 1200; name: 'UCS-2')
, (cp: 1201; name: 'UTF-16BE'), (cp: 1201; name: 'UTF16BE'), (cp:
1201; name: 'UCS-2BE'), (cp: 1201; name: 'unicodeFFFE'), (cp: 1201
; name: 'CP1201'), (cp: 1250; name: 'CP1250'), (cp: 1250; name: 'MS-EE'
), (cp: 1250; name: 'WINDOWS-1250'), (cp: 1250; name: 'windows-1250'
), (cp: 1251; name: 'CP1251'), (cp: 1251; name: 'MS-CYRL'), (cp: 1251
; name: 'WINDOWS-1251'), (cp: 1251; name: 'windows-1251'), (cp: 1252
; name: 'CP1252'), (cp: 1252; name: 'MS-ANSI'), (cp: 1252; name: 'WINDOWS-1252'
```

```
), (cp: 1252; name: 'windows-1252'), (cp: 1253; name: 'CP1253'),
(cp: 1253; name: 'MS-GREEK'), (cp: 1253; name: 'WINDOWS-1253'), (cp
: 1253; name: 'windows-1253'), (cp: 1254; name: 'CP1254'), (cp: 1254
; name: 'MS-TURK'), (cp: 1254; name: 'WINDOWS-1254'), (cp: 1254; name
: 'windows-1254'), (cp: 1255; name: 'CP1255'), (cp: 1255; name: 'MS-HEBR'
), (cp: 1255; name: 'WINDOWS-1255'), (cp: 1255; name: 'windows-1255'
), (cp: 1256; name: 'CP1256'), (cp: 1256; name: 'MS-ARAB'), (cp: 1256
; name: 'WINDOWS-1256'), (cp: 1256; name: 'windows-1256'), (cp: 1257
; name: 'CP1257'), (cp: 1257; name: 'WINBALTRIM'), (cp: 1257; name
: 'WINDOWS-1257'), (cp: 1257; name: 'windows-1257'), (cp: 1258; name
: 'CP1258'), (cp: 1258; name: 'WINDOWS-1258'), (cp: 1258; name: 'windows-1258'
), (cp: 1361; name: 'CP1361'), (cp: 1361; name: 'JOHAB'), (cp: 1361
; name: 'Johab'), (cp: 10000; name: 'macintosh'), (cp: 10001; name
: 'x-mac-japanese'), (cp: 10002; name: 'x-mac-chinesetrad'), (cp:
10003; name: 'x-mac-korean'), (cp: 10004; name: 'x-mac-arabic'),
(cp: 10005; name: 'x-mac-hebrew'), (cp: 10006; name: 'x-mac-greek'
), (cp: 10007; name: 'x-mac-cyrillic'), (cp: 10008; name: 'x-mac-chinesesimp'
), (cp: 10010; name: 'x-mac-romanian'), (cp: 10017; name: 'x-mac-ukrainian'
), (cp: 10021; name: 'x-mac-thai'), (cp: 10029; name: 'x-mac-ce')
, (cp: 10079; name: 'x-mac-icelandic'), (cp: 10081; name: 'x-mac-turkish'
), (cp: 10082; name: 'x-mac-croatian'), (cp: 12000; name: 'UTF-32LE'
), (cp: 12000; name: 'CP12000'), (cp: 12000; name: 'UTF32LE'), (cp
: 12000; name: 'UTF32'), (cp: 12000; name: 'UTF-32'), (cp: 12001;
name: 'UTF-32BE'), (cp: 12001; name: 'CP12001'), (cp: 12001; name
: 'UTF32BE'), (cp: 20000; name: 'x-Chinese_CNS'), (cp: 20001; name
: 'x-cp20001'), (cp: 20002; name: 'x_Chinese-Eten'), (cp: 20003; name
: 'x-cp20003'), (cp: 20004; name: 'x-cp20004'), (cp: 20005; name:
'x-cp20005'), (cp: 20105; name: 'x-IA5'), (cp: 20106; name: 'x-IA5-German'
), (cp: 20107; name: 'x-IA5-Swedish'), (cp: 20108; name: 'x-IA5-Norwegian'
), (cp: 20127; name: 'US-ASCII'), (cp: 20127; name: 'ASCII'), (cp
: 20127; name: 'ANSI_X3.4-1968'), (cp: 20127; name: 'ANSI_X3.4-1986'
), (cp: 20127; name: 'CP367'), (cp: 20127; name: 'IBM367'), (cp: 20127
; name: 'ISO-IR-6'), (cp: 20127; name: 'ISO646-US'), (cp: 20127; name
: 'ISO_646.IRV:1991'), (cp: 20127; name: 'US'), (cp: 20127; name:
'CSASCII'), (cp: 20127; name: 'us-ascii'), (cp: 20261; name: 'x-cp20261'
), (cp: 20269; name: 'x-cp20269'), (cp: 20273; name: 'IBM273'), (cp
: 20277; name: 'IBM277'), (cp: 20278; name: 'IBM278'), (cp: 20280
; name: 'IBM280'), (cp: 20284; name: 'IBM284'), (cp: 20285; name:
'IBM285'), (cp: 20290; name: 'IBM290'), (cp: 20297; name: 'IBM297'
), (cp: 20420; name: 'IBM420'), (cp: 20423; name: 'IBM423'), (cp:
20424; name: 'IBM424'), (cp: 20833; name: 'x-EBCDIC-KoreanExtended'
), (cp: 20838; name: 'IBM-Thai'), (cp: 20866; name: 'koi8-r'), (cp
: 20871; name: 'IBM871'), (cp: 20880; name: 'IBM880'), (cp: 20905
; name: 'IBM905'), (cp: 20924; name: 'IBM00924'), (cp: 20932; name
: 'EUC-JP'), (cp: 20936; name: 'x-cp20936'), (cp: 20949; name: 'x-cp20949'
), (cp: 21025; name: 'cp1025'), (cp: 21866; name: 'koi8-u'), (cp:
28591; name: 'CP819'), (cp: 28591; name: 'IBM819'), (cp: 28591; name
: 'ISO-8859-1'), (cp: 28591; name: 'ISO-IR-100'), (cp: 28591; name
: 'ISO8859-1'), (cp: 28591; name: 'ISO_8859-1'), (cp: 28591; name
: 'ISO_8859-1:1987'), (cp: 28591; name: 'L1'), (cp: 28591; name: 'LATIN1'
), (cp: 28591; name: 'CSISOLATIN1'), (cp: 28591; name: 'iso-8859-1'
), (cp: 28591; name: 'iso8859-1'), (cp: 28592; name: 'iso-8859-2'
), (cp: 28592; name: 'iso8859-2'), (cp: 28593; name: 'iso-8859-3'
), (cp: 28593; name: 'iso8859-3'), (cp: 28594; name: 'iso-8859-4'
```

```
), (cp: 28594; name: 'iso8859-4'), (cp: 28595; name: 'iso-8859-5'
), (cp: 28595; name: 'iso8859-5'), (cp: 28596; name: 'iso-8859-6'
), (cp: 28596; name: 'iso8859-6'), (cp: 28597; name: 'iso-8859-7'
), (cp: 28597; name: 'iso8859-7'), (cp: 28598; name: 'iso-8859-8'
), (cp: 28598; name: 'iso8859-8'), (cp: 28599; name: 'iso-8859-9'
), (cp: 28599; name: 'iso8859-9'), (cp: 28603; name: 'iso-8859-13'
), (cp: 28603; name: 'iso8859-13'), (cp: 28605; name: 'iso-8859-15'
), (cp: 28605; name: 'iso8859-15'), (cp: 29001; name: 'x-Europa')
, (cp: 38598; name: 'iso-8859-8-i'), (cp: 38598; name: 'iso8859-8-i'
), (cp: 50220; name: 'iso-2022-jp'), (cp: 50221; name: 'ISO-2022-JP'
), (cp: 50221; name: 'CP50221'), (cp: 50221; name: 'ISO-2022-JP-MS'
), (cp: 50221; name: 'ISO2022-JP'), (cp: 50221; name: 'ISO2022-JP-MS'
), (cp: 50221; name: 'MS50221'), (cp: 50221; name: 'WINDOWS-50221'
), (cp: 50221; name: 'csISO2022JP'), (cp: 50222; name: 'iso-2022-jp'
), (cp: 50225; name: 'iso-2022-kr'), (cp: 50225; name: 'iso2022-kr'
), (cp: 50227; name: 'x-cp50227'), (cp: 51932; name: 'EUC-JP'), (cp
: 51932; name: 'CP51932'), (cp: 51932; name: 'MS51932'), (cp: 51932
; name: 'WINDOWS-51932'), (cp: 51932; name: 'euc-jp'), (cp: 51936
; name: 'EUC-CN'), (cp: 51949; name: 'euc-kr'), (cp: 52936; name:
'hz-gb-2312'), (cp: 54936; name: 'GB18030'), (cp: 57002; name: 'x-iscii-de'
), (cp: 57003; name: 'x-iscii-be'), (cp: 57004; name: 'x-iscii-ta'
), (cp: 57005; name: 'x-iscii-te'), (cp: 57006; name: 'x-iscii-as'
), (cp: 57007; name: 'x-iscii-or'), (cp: 57008; name: 'x-iscii-ka'
), (cp: 57009; name: 'x-iscii-ma'), (cp: 57010; name: 'x-iscii-gu'
), (cp: 57011; name: 'x-iscii-pa'), (cp: 65001; name: 'UTF-8'), (cp
: 65001; name: 'CP65001'), (cp: 65001; name: 'UTF8'))
```

`UnixCpMap` is a fixed structure with codepage number/codepage name pairs. It is used in GetCodepageData (2174), GetSystemCodepage (2174) and GetCodepageByName (2173) to map code page names to numbers and vice versa.

The map is ordered on code page number, and for equal code page numbers, the names are ordered so the most common one is used first.

```
UnixCpMapLimit = 406 - 83
```

Number of code pages in map `UnixCpMap`.

82.3.2 Types

82.4 Procedures and functions

82.4.1 GetCodepageByName

Synopsis: Find code page by name.

Declaration: `function GetCodepageByName(cpname: RawByteString) : TSystemCodePage`

Visibility: default

Description: `GetCodepageByName` returns the code page number matching `cpname`. The supported code page names are the ones commonly in use in libiconv. Names are searched case-sensitively, with the exception that 'cpN' is converted to 'CPN', where N is a digit.

Errors: If no matching code page name is found, `CP_NONE` is returned.

See also: UnixCpMap (2173), GetSystemCodepage (2174), GetCodepageData (2174)

82.4.2 GetCodepageData

Synopsis: Return index of codepage.

Declaration: `function GetCodepageData(cp: TSystemCodePage) : LongInt`

Visibility: default

Description: `GetCodepageData` returns the index of the first entry in UnixCpMap (2173) which matches `cp`. Since the entries are ordered by code page number, this means the entries can be scanned for alternate code names starting at this index.

Errors: If no matching code page is found, -1 is returned.

See also: UnixCpMap (2173), GetSystemCodepage (2174), GetCodepageByName (2173)

82.4.3 GetSystemCodepage

Synopsis: Return the system code page based on the program environment.

Declaration: `function GetSystemCodepage : TSystemCodePage`

Visibility: default

Description: `GetSystemCodepage` returns the system code page, based on one of the environment variables `LC_ALL`, `LC_CTYPE` or `LANG`. The first non-empty variable (in the order mentioned here) is used.

Errors: If none is found, then a system default is used: Linux and Darwin use `CP_UTF8`, others use `CP_ASCII`.

See also: UnixCpMap (2173), GetSystemCodepage (2174), GetCodepageByName (2173)

82.5 TUnixCpData

```
TUnixCpData = record
  cp : Word;
  name : ansistring;
end
```

`TUnixCpData` contains 2 fields necessary to construct a map between code page number (`cp`) and name (`name`).

Chapter 83

Reference for unit 'unixtype'

83.1 Used units

Table 83.1: Used units by unit 'unixtype'

Name	Page
System	1362

83.2 Overview

The unixtype unit contains the definitions of basic UNIX types. It was initially implemented by Marco van de Voort.

When porting to a new UNIX platform, this unit should be adapted to the sizes and conventions of the platform to which the compiler is ported.

83.3 Constants, types and variables

83.3.1 Constants

```
ARG_MAX = 131072
```

Max number of command-line arguments.

```
NAME_MAX = 255
```

Max length (in bytes) of filename.

```
PATH_MAX = 4095
```

Max length (in bytes) of pathname.

```
Prio_PGrp = 1
```

#rtl.baseunix.fpGetPriority (201) option: Get process group priority.

```
Prio_Process = 0
```

#rtl.baseunix.fpGetPriority (201) option: Get process priority.

```
Prio_User = 2
```

#rtl.baseunix.fpGetPriority (201) option: Get user priority.

```
pthreadrwlocksize = 56
```

Size of `pthread_rwlock_t` _data structure.

```
SIG_MAXSIG = 128
```

Maximum signal number.

```
SYS_NMLN = 65
```

Max system namelength.

```
_PTHREAD_MUTEX_ADAPTIVE_NP = 3
```

Mutex options:.

```
_PTHREAD_MUTEX_DEFAULT = _PTHREAD_MUTEX_NORMAL
```

Mutex options:.

```
_PTHREAD_MUTEX_ERRORCHECK = _PTHREAD_MUTEX_ERRORCHECK_NP
```

Mutex options:.

```
_PTHREAD_MUTEX_ERRORCHECK_NP = 2
```

Mutex options: double lock returns an error code.

```
_PTHREAD_MUTEX_FAST_NP = _PTHREAD_MUTEX_ADAPTIVE_NP
```

Mutex options: Fast mutex.

```
_PTHREAD_MUTEX_NORMAL = _PTHREAD_MUTEX_TIMED_NP
```

Mutex options:.

```
_PTHREAD_MUTEX_RECURSIVE = _PTHREAD_MUTEX_RECURSIVE_NP
```

Mutex options:.

```
_PTHREAD_MUTEX_RECURSIVE_NP = 1
```

Mutex options: recursive mutex.

```
_PTHREAD_MUTEX_TIMED_NP = 0
```

Mutex options: ?

83.3.2 Types

`cbool = longbool`

Boolean type.

`cchar = cint8`

C type: 8-bit signed integer.

`cdouble = Double`

Double precision real format.

`cfloat = single`

Floating-point real format.

`cint = cint32`

C type: integer (natural size).

`cint16 = SmallInt`

C type: 16 bits sized, signed integer.

`cint32 = LongInt`

C type: 32 bits sized, signed integer.

`cint64 = Int64`

C type: 64 bits sized, signed integer.

`cint8 = ShortInt`

C type: 8 bits sized, signed integer.

`clock_t = cuint64`

Clock ticks type.

`clong = Int64`

C type: long signed integer (double sized, depending on platform 32/64 bit).

`clongdouble = extended`

Usually translates to an extended, but is CPU dependent.

`clonglong = cint64`

C type: 64-bit (double long) signed integer.

```
cschar = cint8
```

Signed character type.

```
cshort = cint16
```

C type: short signed integer (half sized).

```
csigned = cint
```

`csigned` is an alias for cint (2177).

```
csint = cint32
```

Signed integer.

```
cslong = Int64
```

The size is CPU dependent.

```
cslonglong = cint64
```

`cslonglong` is an alias for clonglong (2178).

```
csshort = cint16
```

Short signed integer type.

```
cuchar = cuint8
```

C type: 8-bit unsigned integer.

```
cuint = cuint32
```

C type: unsigned integer (natural size).

```
cuint16 = Word
```

C type: 16 bits sized, unsigned integer.

```
cuint32 = LongWord
```

C type: 32 bits sized, unsigned integer.

```
cuint64 = QWord
```

C type: 64 bits sized, unsigned integer.

```
cuint8 = Byte
```

C type: 8 bits sized, unsigned integer.

```
culong = QWord
```

C type: long unsigned integer (double sized,depending on platform 32/64 bit).

```
culonglong = cuint64
```

C type: 64-bit (double long) unsigned integer.

```
cunsigned = cuint
```

Alias for #rtl.unixtype.cuint (2178).

```
cushort = cuint16
```

C type: short unsigned integer (half sized).

```
dev_t = cuint64
```

Device descriptor type.

```
gid_t = cuint32
```

Group ID type.

```
ino64_t = cuint64
```

`ino64_t` is an inode type capable of containing 64-bit inodes.

```
ino_t = clong
```

Inode type.

```
ipc_pid_t = cint
```

Process ID.

```
kDev_t = cushort
```

Kernel device type.

```
mbstate_value_t = record
case Byte of
0: (
  __wch : wint_t;
);
1
  : (
  __wchb : Array[0..3] of Char;
);
end
```

This type should never be used directly. It is part of the mbstate_t (2187) type.

```
mode_t = cint
```

Inode mode type.

```
nlink_t = cuint32
```

Number of links type.

```
off64_t = cint64
```

64-bit offset type.

```
off_t = cint64
```

Offset type.

```
pcbool = ^cbool
```

Pointer to boolean type cbool (2177)

```
pcchar = ^cchar
```

Pointer to #rtl.UnixType.cchar (2177).

```
pcdouble = ^cdouble
```

Pointer to cdouble (2177) type.

```
pcfloat = ^cfloat
```

Pointer to cfloat (2177) type.

```
pcint = ^cint
```

Pointer to cInt (2177) type.

```
pcint16 = ^cint16
```

Pointer to 16-bit signed integer type.

```
pcint32 = ^cint32
```

Pointer to signed 32-bit integer type.

```
pcint64 = ^cint64
```

Pointer to signed 64-bit integer type.

```
pcint8 = ^cint8
```

Pointer to 8-bits signed integer type.

```
pClock = ^clock_t
```

Pointer to TClock (2185) type.

```
pclong = ^clong
```

Pointer to cLong (2177) type.

```
pclongdouble = ^clongdouble
```

Pointer to the long double type clongdouble (2177)

```
pclonglong = ^clonglong
```

Pointer to longlong type.

```
pcschar = ^cschar
```

Pointer to character type cschar (2178).

```
pcshort = ^cshort
```

Pointer to cShort (2178) type.

```
pcsigned = ^csigned
```

Pointer to signed integer type csigned (2178).

```
pcsint = ^csint
```

Pointer to signed integer type csint (2178)

```
pcslong = ^cslong
```

Pointer to the signed long cslong (2178)

```
pcslonglong = ^cslonglong
```

Pointer to Signed longlong type cslonglong (2178)

```
pcsshort = ^csshort
```

Pointer to short signed integer type csshort (2178)

```
pcuchar = ^cuchar
```

Pointer to #rtl.UnixType.cuchar (2178).

```
pcuint = ^cuint
```

Pointer to cUInt (2178) type.

```
pcuint16 = ^cuint16
```

Pointer to 16-bit unsigned integer type.

```
pcuint32 = ^cuint32
```

Pointer to unsigned 32-bit integer type.

```
pcuint64 = ^cuint64
```

Pointer to unsigned 64-bit integer type.

```
pcuint8 = ^cuint8
```

Pointer to 8-bits unsigned integer type.

```
pculong = ^culong
```

Pointer to cuLong (2179) type.

```
pculonglong = ^culonglong
```

Unsigned longlong type.

```
pcunsigned = ^cunsigned
```

Pointer to #rtl.unixtype.cunsigned (2179).

```
pcushort = ^cushort
```

Pointer to cuShort (2179) type.

```
pDev = ^dev_t
```

Pointer to TDev (2185) type.

```
pGid = ^gid_t
```

Pointer to TGid (2185) type.

```
pid_t = cint
```

Process ID type.

```
pIno = ^ino_t
```

Pointer to TIno (2185) type.

```
pIno64 = ^ino64_t
```

Pointer to ino64_t (2179).

```
pkDev = ^kDev_t
```

Pointer to TkDev (2186) type.

```
pmbstate_t = ^mbstate_t
```

Pointer to mbstate_t (2187) type.

```
pMode = ^mode_t
```

Pointer to TMode (2186) type.

```
pnLink = ^nlink_t
```

Pointer to TnLink (2186) type.

```
pOff = ^off_t
```

Pointer to TOff (2186) type.

```
pOff64 = ^off64_t
```

Pointer to `off64_t` type.

```
pPid = ^pid_t
```

Pointer to TPid (2186) type.

```
pSize = ^size_t
```

Pointer to TSize (2186) type.

```
psize_t = pSize
```

Pointer to size_t (2185) type.

```
pSockLen = ^socklen_t
```

Pointer to TSockLen (2186) type.

```
pSSize = ^ssize_t
```

Pointer to TsSize (2186) type.

```
PStatFS = ^TStatfs
```

Pointer to TStatFS (2189) type.

```
pthread_key_t = cuint
```

Thread local storage key (opaque).

```
PTHREAD_MUTEX_T = record
case Byte of
1: (
  __m_reserved : LongInt
  ;
  __m_count : LongInt;
  __m_owner : pointer;
  __m_kind : LongInt
  ;
  __m_lock : record
    __status : SizeInt;
    __spinlock : LongInt
  ;
  end;
);
end
```

_pthread_mutex_t describes a thread mutex. It should be considered an opaque record, the names of the fields can change anytime.

```
pthread_rwlock_t = record
case Boolean of
False: (
  _data : Array
  [0..pthreadrwlocksize-1] of Char;
);
True: (
  align : clong;
);
end
```

pthread_rwlock_t describes a lock. It should be considered an opaque record, the names of the fields can change anytime.

```
pthread_t = culong
```

Thread description record.

```
pTime = ^time_t
```

Pointer to TTime (2186) type.

```
ptimespec = ^timespec
```

Pointer to timespec (2189) record.

```
ptimeval = ^timeval
```

Pointer to timeval (2189) record.

```
ptime_t = ^time_t
```

Pointer to time_t (2185) type.

```
pUid = ^uid_t
```

Pointer to TUid (2186) type.

```
pwchar_t = ^wchar_t
```

Pointer to wchar_t (2187) type.

```
size_t = cuint64
```

Size specification type.

```
socklen_t = cuint32
```

Socket address length type.

```
ssize_t = cint64
```

Small size type.

```
TClock = clock_t
```

Alias for clock_t (2177) type.

```
TDev = dev_t
```

Alias for dev_t (2179) type.

```
TGid = gid_t
```

Alias for gid_t (2179) type.

```
time_t = cint64
```

Time span type.

```
TIno = ino_t
```

Alias for ino_t (2179) type.

```
TIno64 = ino64_t
```

Alias for ino64_t (2179).

```
TIOCtlRequest = culong
```

Opaque type used in FpIOCtl (202).

`TkDev = kDev_t`

Alias for kDev_t (2179) type.

`TMode = mode_t`

Alias for mode_t (2180) type.

`TnLink = nlink_t`

Alias for nlink_t (2180) type.

`TOff = off_t`

Alias for off_t (2180) type.

`TOff64 = off64_t`

Alias for `off64_t` type.

`TPid = pid_t`

Alias for pid_t (2182) type.

`TSize = size_t`

Alias for size_t (2185) type.

`TSockLen = socklen_t`

Alias for socklen_t (2185) type.

`TSSize = ssize_t`

Alias for ssize_t (2185) type.

`TTime = time_t`

Alias for TTime (2186) type.

`TTimeSpec = timespec`

Alias for TimeSpec (2189) type.

`TTimeVal = timeval`

Alias for TimeVal (2189) record.

`TUid = uid_t`

Alias for uid_t (2187) type.

```
uid_t = cuint32
```

User ID type.

```
wchar_t = cint32
```

Wide character type.

```
wint_t = cint32
```

Wide character size type.

83.4 mbstate_t

```
mbstate_t = record
  __count : cint;
  __value : mbstate_value_t;
  end
```

This type should never be used directly.

83.5 pthread_attr_t

```
pthread_attr_t = record
  __detachstate : cint;
  __schedpolicy :
  cint;
  __schedparam : sched_param;
  __inheritsched : cint;
  __scope
  : cint;
  __guardsize : size_t;
  __stackaddr_set : cint;
  __stackaddr
  : pointer;
  __stacksize : size_t;
end
```

`pthread_attr_t` describes the thread attributes. It should be considered an opaque record, the names of the fields can change anytime. Use the appropriate functions to set the thread attributes.

83.6 pthread_condattr_t

```
pthread_condattr_t = record
  __dummy : cint;
end
```

`pthread_condattr_t` describes the attributes of a thread mutex. It should be considered an opaque record, the names of the fields can change anytime.

83.7 pthread_cond_t

```
pthread_cond_t = record
  __c_lock : _pthread_fastlock;
  __c_waiting
  : pointer;
  __padding : Array[0..48-1-sizeof(_pthread_fastlock)
  -sizeof(pointer)-sizeof(clonglong)] of Byte;
  __align : clonglong
  ;
end
```

pthread_cond_t describes a thread conditional variable. It should be considered an opaque record, the names of the fields can change anytime.

83.8 pthread_mutexattr_t

```
pthread_mutexattr_t = record
  __mutexkind : cint;
end
```

pthread_mutexattr_t describes the attributes of a thread mutex. It should be considered an opaque record, the names of the fields can change anytime.

83.9 pthread_rwlockattr_t

```
pthread_rwlockattr_t = record
  __lockkind : cint;
  __pshared : cint
  ;
end
```

pthread_rwlockattr_t describes the attributes of a lock. It should be considered an opaque record, the names of the fields can change anytime.

83.10 sched_param

```
sched_param = record
  __sched_priority : cint;
end
```

Scheduling parameter description record.

83.11 sem_t

```
sem_t = record
```

```
  __sem_lock : _pthread_fastlock;
  __sem_value : cint
  ;
  __sem_waiting : pointer;
end
```

sem_t describes a thread semaphore. It should be considered an opaque record, the names of the fields can change anytime.

83.12 timespec

```
timespec = record
  tv_sec : time_t;
  tv_nsec : clong;
end
```

Record specifying time interval.

83.13 timeval

```
timeval = record
  tv_sec : time_t;
  tv_usec : clong;
end
```

Time specification type.

83.14 TStatfs

```
TStatfs = record
  fstype : clong;
  bsize : clong;
  blocks : culong
  ;
  bfree : culong;
  bavail : culong;
  files : culong;
  ffree
  : culong;
  fsid : Array[0..1] of cint;
  namelen : clong;
  frsize
  : clong;
  flags : clong;
  spare : Array[0..3] of clong;
end
```

Record describing a file system in the unix.fpstatfs (2175) call.

83.15 _pthread_fastlock

```
_pthread_fastlock = record
  __status : clong;
  __spinlock : cint
  ;
end
```

`_pthread_fastlock` describes a thread mutex. It should be considered an opaque record, the names of the fields can change anytime.

Chapter 84

Reference for unit 'unixutil'

84.1 Used units

Table 84.1: Used units by unit 'unixutil'

Name	Page
BaseUnix	146
System	1362

84.2 Overview

The UnixUtil unit contains some of the routines that were present in the old Linux unit, but which do not really belong in the UNIX (2135) or baseunix (146) units.

Most of the functions described here have cross-platform counterparts in the SysUtils (1635) unit. It is therefore recommended to use that unit.

84.3 Procedures and functions

84.3.1 ArrayStringToPPchar

Synopsis: Concert an array of string to an array of null-terminated strings.

Declaration:
```
function ArrayStringToPPchar(const S: Array of RawByteString;
                             reserveentries: LongInt) : PPChar
```

Visibility: default

Description: `ArrayStringToPPchar` creates an array of null-terminated strings that point to strings which are the same as the strings in the array `S`. The function returns a pointer to this array. The array and the strings it contains must be disposed of after being used, because it they are allocated on the heap.

The `ReserveEntries` parameter tells `ArrayStringToPPchar` to allocate room at the end of the array for another `ReserveEntries` entries.

Errors: If not enough memory is available, an error may occur.

See also: StringToPPChar (2192)

84.3.2 StringToPPChar

Synopsis: Split string in list of null-terminated strings.

Declaration:
```
function StringToPPChar(S: PChar; ReserveEntries: Integer) : PPChar
function StringToPPChar(var S: RawByteString; ReserveEntries: Integer)
                        : PPChar
```

Visibility: default

Description: `StringToPPChar` splits the string `S` in words, replacing any whitespace with zero characters. It returns a pointer to an array of pchars that point to the first letters of the words in `S`. This array is terminated by a `Nil` pointer.

The function does *not* add a zero character to the end of the string unless it ends on whitespace.

The function reserves memory on the heap to store the array of `PChar`; The caller is responsible for freeing this memory.

This function can be called to create arguments for the various `Exec` calls.

Errors: None.

See also: ArrayStringToPPchar (2191), #rtl.baseunix.FpExecve (192)

Listing: ./examples/unutilex/ex70.pp

```
Program Example70;

{ Program to demonstrate the StringToPPchar function. }

Uses UnixUtil;

Var S : String;
    P : PPChar;
    I : longint;

begin
  // remark whitespace at end.
  S:='This is a string with words. ';
  P:=StringToPPChar(S,0);
  I:=0;
  While P[i]<>Nil do
    begin
    Writeln('Word ',i,' : ',P[i]);
    Inc(I);
    end;
  FreeMem(P,i*SizeOf(Pchar));
end.
```

Chapter 85

Reference for unit 'Variants'

85.1 Used units

Table 85.1: Used units by unit 'Variants'

Name	Page
rtlconsts	??
sysconst	??
System	1362
sysutils	1635
TypInfo	2016

85.2 Overview

The compiler has built-in support for variants, and for many operations, variants can be used without thinking about it. The system unit has built-in support for some of the basic operations on a variant, as well as some compiler helper routines. However, some operations and definitions are implemented in the `Variants` unit so as not to burden the system unit with routines that may not always be needed.

There is a basic set of variants that are defined by the Windows OS, these are supported by the compiler and the routines in the `Variants` unit. Additional variant types can be registered using the TCustomVariantType (2222) type.

The variants unit also registers a handler for setting published properties (using RTTI) using variant-typed values. Nothing needs to be done for this except including the variants unit in your program.

85.3 Constants, types and variables

85.3.1 Constants

```
CFirstUserType = CMinVarType + CIncVarType
```

`CFirstUserType` is the first allocated value for `vType` when registering a variant type by the RTL, when instantiating a TCustomVariantType (2222) to register a variant type. It is better not to specify a custom variant type value, but let the system allocate a custom variant type.

```
CIncVarType = $000F
```

CIncVarType specifies the width of the gap after CMinVarType (2194). No user types are registered between CMinVarType and CMinVarType+CIncVarType.

```
CMaxNumberOfCustomVarTypes = $0EFF
```

CMaxNumberOfCustomVarTypes is the amount of custom variant types that can be registered. The range of the variant type indicator (vType) only allows a limited amount of variants. This constant is the maximum amount.

```
CMaxVarType = CMinVarType + CMaxNumberOfCustomVarTypes
```

CMaxVarType is the maximum allowed value for vType when registering a variant type, instantiating a TCustomVariantType (2222) to register a variant type. It is better not to specify a value, but let the system allocate a custom variant type.

```
CMinVarType = $0100
```

CMinVarType is the first allowed value for vType when registering a variant type by the OS, instantiating a TCustomVariantType (2222) to register a variant type. It is better not to specify a custom variant type value, but let the system allocate a custom variant type.

```
FloatVarTypes = [varSingle, varDouble, varCurrency, varDecimal]
```

FloatlVarTypes is used in VarIsFloat (2211) to decide which variant types are considered ordinals.

```
OrdinalVarTypes = [varSmallInt, varInteger, varBoolean, varShortInt
  , varByte, varWord, varLongWord, varInt64, varQWord]
```

OrdinalVarTypes is used in VarIsOrdinal (2211) to decide which variant types are considered ordinals.

```
VarOpAsText : Array[TVarOp] of string = ('+', '-', '*', '/', 'div'
  , 'mod', 'shl', 'shr', 'and', 'or', 'xor', '', '-', 'not', '=', '<>'
  , '<', '<=', '>', '>=', '**')
```

VarOpAsText is an array with the names of the various variant operations.

85.3.2 Types

```
TAnyProc = procedure(var V: tvardata)
```

TAnyProc is the type for the ClearAnyProc (2196), ChangeAnyProc (2196) and RefAnyProx (2193) callbacks. It accepts a reference to a variant record. The operation to be performed depends on the callback.

```
TBooleanToStringRule = (bsrAsIs,bsrLower,bsrUpper)
```

Table 85.2: Enumeration values for type TBooleanToStringRule

Value	Explanation
bsrAsIs	Leave casing as is.
bsrLower	Convert to lowercase.
bsrUpper	Convert to uppercase.

`TBooleanToStringRule` describes how boolean values are converted to string values. It is defined for Delphi compatibility, but is not used in the FPC runtime.

```
TCustomVariantTypeClass = Class of TCustomVariantType
```

`TCustomVariantTypeClass` is the class type of `TCustomVariantType`.

```
TNullCompareRule = (ncrError,ncrStrict,ncrLoose)
```

Table 85.3: Enumeration values for type TNullCompareRule

Value	Explanation
ncrError	Raise an error when one of the values is Null.
ncrLoose	Attempt to compare anyway.
ncrStrict	Act as if the comparison is false.

`TNullCompareRule` is the type for the NullEqualityRule (2197) and NullMagnitudeRule (2197) variables. It can have the following values:

ncrError Raise an error when one of the values is Null.

ncrStrict Act as if the comparison is false.

ncrLoose Attempt to compare anyway.

```
TVarCompareResult = (crLessThan,crEqual,crGreaterThan)
```

Table 85.4: Enumeration values for type TVarCompareResult

Value	Explanation
crEqual	Both values are equal.
crGreaterThan	The first value is greater than the second value.
crLessThan	The first value is less than the second value.

`TVarCompareResult` is used when comparing 2 custom variant values in TCustomVariantType.Compare (2226). The following values exist:

crLessThan The first value is less than the second value.

crGreaterThan The first value is greater than the second value.

crEqual Both values are equal.

```
TVarDataArray = Array of tvardata
```

`TVarDataArray` is a helper type for instance used in IVarInvokeable.DoFunction (2221) to represent the arguments passed to a function.

```
TVarDispProc = procedure(Dest: PVariant; const Source: Variant;
  CallDesc: pcalldesc; Params: Pointer)
```

`TVarDispProc` is the type for the VarDispProc (2198) callback handler. It accepts a destination variant (`Dest`) for a result. `Source` is the variant on which the operation was invoked, `CallDesc` Describes the arguments to the call and `Params` points to the parameters provided in the call.

```
TVariantRelationship = (vrEqual,vrLessThan,vrGreaterThan,vrNotEqual
  )
```

Table 85.5: Enumeration values for type TVariantRelationship

Value	Explanation
vrEqual	Are the 2 variants equal.
vrGreaterThan	Is the first variant (strictly) greater than the second.
vrLessThan	Is the first variant (strictly) less than the second.
vrNotEqual	Are the 2 variants unequal.

`TVariantRelationship` is used by VarCompareValue (2207) to indicate the type of comparison operation it must perform. It has the following values:

vrEqual Are the 2 variants equal.

vrLessThan Is the first variant (strictly) less than the second.

vrGreaterThan Is the first variant (strictly) greater than the second.

vrEqual Are the 2 variants equal.

vrEqual Are the 2 variants equal.

85.3.3 Variables

```
ChangeAnyProc : TAnyProc
```

`ChangeAnyProc` is currently not used in the `Variants` unit.

```
ClearAnyProc : TAnyProc
```

`ClearAnyProc` is called when the system needs to clear a variant of type `varAny`. it must clear the variant.

```
EmptyParam : OleVariant
```

EmptyParam is an initialized variant with type varError and error value VAR_PARAMNOTFOUND. In difference with Null (2200) and UnAssigned (2201) it is a variable. You should take care never to write to it.

```
InvalidCustomVariantType : TCustomVariantType
```

InvalidCustomVariantType can be set to a TCustomVariantType (2222) instance to indicate an an unknown type. By default it is set to Pointer(-1).

```
NullAsStringValue : string = ''
```

NullAsStringValue is the value used when converting a Null to a string. It is only used when NullStrictConvert (2197) is False, if NullStrictConvert (2197) is True, the value of NullAsStringValue is ignored.

```
NullEqualityRule : TNullCompareRule = ncrLoose
```

NullEqualityRule is checked when the system needs to compare the equality of variants with Null values (operations opCmpEq, opCmpNe). Check TNullCompareRule (2195) for a list of allowed values. The default is ncrLoose.

For determining the order (less than, greater than etc.) see NullMagnitudeRule (2197).

```
NullMagnitudeRule : TNullCompareRule = ncrLoose
```

NullEqualityRule is checked when the system needs to determine the ordering of variants with Null values (operations like opCmpGe, opCmpLe). Check TNullCompareRule (2195) for a list of allowed values. The default is ncrLoose.

For determining equality (opCmpEq, opCmpNe) of Null values, see NullEqualityRule (2197).

```
NullStrictConvert : Boolean = True
```

NullStrictConvert determines what to do when typecasting a Null value to another type: If it is True then an exception will be raised using VarCastError (2206). If it is False then a sensible default is used: 0 or some variation on 0 based on the type of the variant. For string values the NullAsStringValue (2197) is used.

```
OleVariantInt64AsDouble : Boolean = False
```

OleVariantInt64AsDouble describes what to do when a Int64 value must be converted to a OleVariant value. When True the Int64 variant is cast to a double. When False, it remains an int64 value.

```
PackVarCreation : Boolean = True
```

PackVarCreation determines what to do when a variant array is created for elements with an integer type (varSmallint, varByte and the like). A value of False means the array will contain varInteger elements. This can result in better memory alignment. When PackVarCreation is True then the smallest possible size is selected for the elements.

```
RefAnyProc : TAnyProc
```

RefAnyProc is called when the system needs to obtain a reference to a variant of type varAny. it must replace the variant with a reference to the variant.

```
VarDispProc : TVarDispProc
```

VarDispProc is the handler invoked when a dispatch invoke is handled on a variant of type varDispatch, varAny or varUnknown.

85.4 Procedures and functions

85.4.1 DynArrayFromVariant

Synopsis: Convert a variant to a dynamic arrayt.

Declaration:
```
procedure DynArrayFromVariant(var DynArray: Pointer; const V: Variant;
                              TypeInfo: Pointer)
```

Visibility: default

Description: DynArrayFromVariant transforms a variant array to a dynamic array. It uses TypInfo to calculate the length, element type and dimension of the array.

The opposite transformation can be performed with DynArrayToVariant (2198).

See also: VarArrayCreateError (2202), VarArrayCreate (2201), VarArrayOf (2204), DynArrayToVariant (2198)

85.4.2 DynArrayToVariant

Synopsis: Convert a Dynamic Array To a Variant.

Declaration:
```
procedure DynArrayToVariant(var V: Variant; const DynArray: Pointer;
                            TypeInfo: Pointer)
```

Visibility: default

Description: DynArrayToVariant converts the dynamic array DynArray to a variant array V. It uses the type information in TypeInfo to calculate the number of dimensions, array lengths and type of the element. The dynamic array can only contain basic types.

If there is no data, an empty variant will be returned.

The opposite transformation can be performed with DynArrayFromVariant (2198).

See also: VarArrayCreateError (2202), VarArrayCreate (2201), VarArrayOf (2204), DynArrayFromVariant (2198)

85.4.3 FindCustomVariantType

Synopsis: Find a custom variant class on vartype.

Declaration:
```
function FindCustomVariantType(const aVarType: tvartype;
                               out CustomVariantType: TCustomVariantType)
                                : Boolean;  Overload
function FindCustomVariantType(const TypeName: string;
                               out CustomVariantType: TCustomVariantType)
                                : Boolean;  Overload
```

Visibility: default

Description: `FindCustomVariantType` searches the registry of known TCustomVariantType (2222) classes and returns the instance registered for the variant type `aVarType` or the class name `TypeName` in `CustomVariantType` if found.

It returns `True` if it found a matching definition, or `False` otherwise.

See also: TCustomVariantType (2222)

85.4.4 FindVarData

Synopsis: Return a pointer to variant data.

Declaration:
```
function FindVarData(const V: Variant) : pvardata
```

Visibility: default

Description: `FindVarData` returns a pointer to the argument `V` if it is not a reference. If it is a reference, then the reference pointer is returned.

85.4.5 GetPropValue

Synopsis: Return a property value as a variant.

Declaration:
```
function GetPropValue(Instance: TObject; PropInfo: PPropInfo;
                      PreferStrings: Boolean) : Variant;  Overload
```

Visibility: default

Description: `GetPropValue` returns the value of the property described by `PropInfo` from the `Instance`. The property value is returned as a variant type.

when `PreferStrings` is true, the implementation will tend to cast to a string-valued variant when a conversion must be done.

This function is used as the value for the TypInfo (2193) unit's OnGetPropValue (2193) callback. This callback is automatically initialized with the function when the variants unit is used.

See also: TypInfo (2193), TypInfo.OnGetPropValue (2193), SetPropValue (2200)

85.4.6 GetVariantProp

Synopsis: Get variant valued property.

Declaration:
```
function GetVariantProp(Instance: TObject; PropInfo: PPropInfo)
                        : Variant
function GetVariantProp(Instance: TObject; const PropName: string)
                        : Variant
```

Visibility: default

Description: `GetVariantProp` returns the value of the variant-types property described by `PropInfo` or `PropName` from the `Instance`.

This function is used as the value for the TypInfo (2193) unit's OnGetVariantProp (2193) callback. This callback is automatically initialized with the function when the variants unit is used.

See also: TypInfo (2193), TypInfo.OnGetVariantProp (2193), SetVariantProp (2201), GetPropValue (2199)

85.4.7 HandleConversionException

Synopsis: Convert an exception to a variant exception.

Declaration:
```
procedure HandleConversionException(const ASourceType: tvartype;
                                    const ADestType: tvartype)
```

Visibility: default

Description: `HandleConversionException` converts a RTL exception (EConvertError (2193) or ERangeError (2193) or EOverFlow (2193)) to an appropriate variant error (varCastError (2206) and varOverFlowError (2213)). Other exceptions are re-raised. If a source and destination type `ASourceType` and `aDestType` are specified they are included in the error message.

See also: EConvertError (2193), ERangeError (2193), varCastError (2206), varOverFlowError (2213), EOverFlow (2193)

85.4.8 Null

Synopsis: Return a null variant.

Declaration:
```
function Null : Variant
```

Visibility: default

Description: `UnAssigned` returns a Null variant (`type = varNull`). It can be used to test for equality with a Null variant.

See also: UnAssigned (2201), EmptyParam (2197)

85.4.9 SetClearVarToEmptyParam

Synopsis: Create an error variant with value `VAR_PARAMNOTFOUND`.

Declaration:
```
procedure SetClearVarToEmptyParam(var V: tvardata)
```

Visibility: default

Description: `SetClearVarToEmptyParam` clears the variant `Vvar>` and sets it's type to `varError` and value to `VAR_PARAMNOTFOUND`. `VarIsEmptyParam(SetClearVarToEmptyParam(V))` will return `True`.

Errors: None.

See also: VarIsEmptyParam (2210), VarIsError (2210)

85.4.10 SetPropValue

Synopsis: Set a property value as a variant.

Declaration:
```
procedure SetPropValue(Instance: TObject; PropInfo: PPropInfo;
                       const Value: Variant);  Overload
```

Visibility: default

Description: `SetPropValue` sets the value of the property described by `PropInfo` from the `Instance`. The property value is set from the variant-typed value `Value`.

This function is used as the value for the TypInfo (2193) unit's OnSetPropValue (2193) callback. This callback is automatically initialized with the function when the variants unit is used.

See also: TypInfo (2193), TypInfo.OnSetPropValue (2193), GetPropValue (2199)

85.4.11 SetVariantProp

Synopsis: Set variant valued property.

Declaration:
```
procedure SetVariantProp(Instance: TObject; const PropName: string;
                         const Value: Variant)
procedure SetVariantProp(Instance: TObject; PropInfo: PPropInfo;
                         const Value: Variant)
```

Visibility: default

Description: `SetVariantProp` sets the value of the variant-types property described by `PropInfo` or `PropName` from the `Instance` to `Value`.

This function is used as the value for the TypInfo (2193) unit's OnSetVariantProp (2193) callback. This callback is automatically initialized with the function when the variants unit is used.

See also: TypInfo (2193), TypInfo.OnSetVariantProp (2193), GetVariantProp (2199), GetPropValue (2199)

85.4.12 Unassigned

Synopsis: Return an unassigned variant.

Declaration:
```
function Unassigned : Variant
```

Visibility: default

Description: `UnAssigned` returns an unassigned variant (`type = varEmpty`). It can be used to test for equality with an empty variant.

See also: Null (2200), EmptyParam (2197)

85.4.13 VarArrayAsPSafeArray

Synopsis: Return internal array of variant value.

Declaration:
```
function VarArrayAsPSafeArray(const A: Variant) : pvararray
```

Visibility: default

Description: `VarArrayAsPSafeArray` returns the internal array of the variant `A` if it is a variant array. If not, an exception is raised.

Errors: if the variant `A` is not an array, an EVariantInvalidArgError (2218) exception is raised.

See also: VarIsArray (2209), EVariantInvalidArgError (2218)

85.4.14 VarArrayCreate

Synopsis: Create a variant array.

Declaration:
```
function VarArrayCreate(const Bounds: Array of SizeInt;
                        aVarType: tvartype) : Variant
function VarArrayCreate(const Bounds: pvararrayboundarray;
                        Dims: SizeInt; aVarType: tvartype) : Variant
```

Visibility: default

Description: `VarArrayCreate` creates a (optionally multidimensional) array with upper,lower bounds specified in `Bounds`. The number of bounds (in case of a single array) must be even: 2 bounds for every dimension of the array are required. All elements of the array are of the same type. The following examples create a one-dimensional array with 10 elements

```
VarArrayCreate([0,9],varInteger);
 VarArrayCreate([1,10],varInteger);
```

The first array is 0-based, the second is 1-based. The following creates a 2-dimensional array:

```
VarArrayCreate([0,9,0,1],varInteger);
 VarArrayCreate([1,10,1,2],varInteger);
```

The first array is 0-based, the second is 1-based. Each array consists of an array of 2 elements.

The array can also be specified as a pointer to array of system.tvararraybound (2193) records, and a number of dimensions. The above 1-dimensional arrays can be specified as:

```
var
  B : tvararraybound;
begin
  b.elementcount:=10;
  B.lowbound:=0;
  VarArrayCreate(@B,1,varInteger);
  b.elementcount:=10;
  B.lowbound:=1;
  VarArrayCreate(@B,1,varInteger);
```

Errors: If an uneven amount of bounds is specified or the operating system failed to create the array, an exception is raised using VarArrayCreateError (2202)

See also: VarArrayCreateError (2202), VarArrayOf (2204)

85.4.15 VarArrayCreateError

Synopsis: Raise an `EVariantArrayCreateError` error.

Declaration: `procedure VarArrayCreateError`

Visibility: default

Description: `VarArrayCreateError` raises an EVariantArrayCreateError (2217) exception with a standard error message.

See also: EVariantArrayCreateError (2217), VarOverflowError (2213), VarInvalidNullOp (2208), VarInvalidOp (2208), VarBadIndexError (2205), VarArrayLockedError (2203)

85.4.16 VarArrayDimCount

Synopsis: Return the number of dimensions of the array.

Declaration: `function VarArrayDimCount(const A: Variant) : LongInt`

Visibility: default

Description: `VarArrayDimCount` returns the number of dimensions of the array `A`. If `A` is not an array, zero is returned.

Errors: None.

See also: VarArrayCreate (2201), VarArrayLowBound (2204), VarArrayHighBound (2203)

85.4.17 VarArrayHighBound

Synopsis: Return lower bound of an array.

Declaration: `function VarArrayHighBound(const A: Variant; Dim: LongInt) : LongInt`

Visibility: default

Description: `VarArrayHighBound` returns the high bound (max index) of dimension `Dim` of array `A`. The dimension `Dim` is 1-based.

Errors: If the dimension is out of range, or `A` is not a variant array, an exception will be raised.

See also: VarArrayCreate (2201), VarArrayDimCount (2202), VarArrayLowBound (2204)

85.4.18 VarArrayLock

Synopsis: get a pointer to data of a variant array.

Declaration: `function VarArrayLock(const A: Variant) : Pointer`

Visibility: default

Description: `VarArrayLock` returns a pointer to the data of an array-typed variant `A` and locks the variant. The pointer can then be used to manipulate the data of the array and be sure that the data is not modified by another process or thread. While the data is not released using VarArrayUnlock (2204) the variant cannot be changed or released, it is therefor impperative that the variant is again released. Memory leaks will be the result if this is not done.

See also: VarArrayUnlock (2204)

85.4.19 VarArrayLockedError

Synopsis: Raise an `EVariantArrayLockedError` error.

Declaration: `procedure VarArrayLockedError`

Visibility: default

Description: `VarArrayLockedError` raises an EVariantArrayLockedError (2217) exception with a standard error message.

See also: EVariantArrayLockedError (2217), VarInvalidNullOp (2208), VarInvalidOp (2208), VarBadIndex-Error (2205)

85.4.20 VarArrayLowBound

Synopsis: Return lower bound of an array.

Declaration: `function VarArrayLowBound(const A: Variant; Dim: LongInt) : LongInt`

Visibility: default

Description: `VarArrayLowBound` returns the lower bound (min index) of dimension `Dim` of array `A`. The dimension `Dim` is 1-based.

Errors: If the dimension is out of range, or `A` is not a variant array, an exception will be raised.

See also: VarArrayCreate (2201), VarArrayDimCount (2202), VarArrayHighBound (2203)

85.4.21 VarArrayOf

Synopsis: Create a variants array of a series of values.

Declaration: `function VarArrayOf(const Values: Array of Variant) : Variant`

Visibility: default

Description: `VarArrayOf` creates a variant array with elements of type `varVariant`. The array has as many values as there are elements in `Values` and the element values are copied from `Values`.

Errors: If the array cannot be created, an `EVariantError` exception may be raised.

See also: VarArrayCreateError (2202), VarArrayCreate (2201), DynArrayToVariant (2198)

85.4.22 VarArrayRef

Synopsis: Get a reference to a variant array.

Declaration: `function VarArrayRef(const A: Variant) : Variant`

Visibility: default

Description: `VarArrayRef` returns a reference to the variant array `A`.

Errors: An exception will be raised if the variant `A` is not an array.

See also: VarArrayLock (2203), VarArrayUnlock (2204), VarIsArray (2209)

85.4.23 VarArrayUnlock

Synopsis: Release data captured by `VarArrayLock`.

Declaration: `procedure VarArrayUnlock(const A: Variant)`

Visibility: default

Description: `VarArrayUnlock` unlocks the variant array `A` that was previously locked using VarArrayLock (2203).

See also: VarArrayLock (2203)

85.4.24 VarAsError

Synopsis: Create an error-typed variant.

Declaration: `function VarAsError(AResult: HRESULT) : Variant`

Visibility: default

Description: `VarAsError` creates an error-typed variant with value `aResult`.

Errors: None.

See also: SetClearVarToEmptyParam (2200), VarIsError (2210)

85.4.25 VarAsType

Synopsis: Attempt to cast a variant to another type.

Declaration: `function VarAsType(const V: Variant; aVarType: tvartype) : Variant`

Visibility: default

Description: `VarAsType` attempts to cast the variant `V` to a new variant of type `aVarType` and returns the new variant.

Errors: If the requested type cast is not supported or possible, an EVariantTypeCastError (2220) exception may be raised.

See also: EVariantTypeCastError (2220), VarIsType (2212), VarAsType (2205)

85.4.26 VarBadIndexError

Synopsis: Raise an `EVariantBadIndexError` error.

Declaration: `procedure VarBadIndexError`

Visibility: default

Description: `VarBadIndexError` raises an EVariantBadIndexError (2218) exception with a standard error message.

See also: EVariantBadIndexError (2218), VarInvalidNullOp (2208), VarInvalidOp (2208), VarArrayLockedError (2203)

85.4.27 VarBadTypeError

Synopsis: Raise an `EVariantBadVarTypeError` error.

Declaration: `procedure VarBadTypeError`

Visibility: default

Description: `VarBadTypeError` raises an EVariantBadVarTypeError (2218) exception.

See also: EVariantBadVarTypeError (2218), VarInvalidNullOp (2208), VarInvalidOp (2208)

85.4.28 VarCastError

Synopsis: Raise a variant type cast error `EVariantTypeCastError`.

Declaration:
```
procedure VarCastError
procedure VarCastError(const ASourceType: tvartype;
                       const ADestType: tvartype)
```

Visibility: default

Description: `VarCastError` raises an EVariantTypeCastError (2220) exception. If the source and destination types are specified, then a description of the types is included in the error message.

See also: EVariantTypeCastError (2220), VarCastErrorOle (2206)

85.4.29 VarCastErrorOle

Synopsis: Raise a variant type cast error `EVariantTypeCastError`.

Declaration:
```
procedure VarCastErrorOle(const ASourceType: tvartype)
```

Visibility: default

Description: `VarCastError` raises an EVariantTypeCastError (2220) exception. a description of the source type (`aSourceType`) is included in the error message.

See also: EVariantTypeCastError (2220), VarCastError (2206)

85.4.30 VarCheckEmpty

Synopsis: Raise exception if a variant is empty.

Declaration:
```
procedure VarCheckEmpty(const V: Variant)
```

Visibility: default

Description: `VarCheckEmpty` will raise an EVariantError (2193) exception if the variant `V` is empty (it uses VarIsEmpty (2210) to check this.)

Errors: if the variant is empty an EVariantError (2193) exception is raised.

See also: VarIsEmpty (2210), VarType (2216), VarIsNull (2211)

85.4.31 VarClear

Synopsis: Clear the variant value.

Declaration:
```
procedure VarClear(var V: Variant)
procedure VarClear(var V: OleVariant)
```

Visibility: default

Description: `VarClear` clears the variant, possibly freeing any memory taken by the value

Errors: None.

See also: SetClearVarToEmptyParam (2200), VarIsError (2210), VarIsClear (2209)

85.4.32 VarCompareValue

Synopsis: Compare 2 variant values.

Declaration:
```
function VarCompareValue(const A: Variant; const B: Variant)
                                 : TVariantRelationship
```

Visibility: default

Description: `VarCompareValue` compares 2 variants `A` and `B`. It returns one of the following values:

vrEqual if the 2 variant values are equal.

vrUnEqual if one of the 2 variant is null or empty and the other is not.

vrGreaterThan if `A>B`

vrLessThan if `A<B`

See also: VarSameValue (2213)

85.4.33 VarCopyNoInd

Synopsis: Not supported.

Declaration:
```
procedure VarCopyNoInd(var Dest: Variant; const Source: Variant)
```

Visibility: default

Description: `VarCopyNoInd` is currently not supported.

Errors: An `EVariantError` exception is always raised.

85.4.34 VarEnsureRange

Synopsis: Make sure the variant is within a specified range.

Declaration:
```
function VarEnsureRange(const AValue: Variant; const AMin: Variant;
                              const AMax: Variant) : Variant
```

Visibility: default

Description: `VarEnsureRange` checks `AValue` and returns it if it is in the range specified by `AMin`, `AMax`. If it is less than `AMin`, then `AMin` is returned. If it is larger than `AMax`, then `AMax` is returned.

Errors: If the variants are of different types and they cannot be converted, an exception will be raised.

See also: VarInRange (2208)

85.4.35 VarFromDateTime

Synopsis: Create variant from `TDateTime` value.

Declaration:
```
function VarFromDateTime(const DateTime: TDateTime) : Variant
```

Visibility: default

Description: `VarFromDateTime` creates a variant with type `varDate` and value `DateTime`.

Errors: None.

See also: varToDateTime (2214)

85.4.36 VarInRange

Synopsis: Check if a variant is in a range of values.

Declaration:
```
function VarInRange(const AValue: Variant; const AMin: Variant;
                    const AMax: Variant) : Boolean
```

Visibility: default

Description: `VarInRange` is an auxiliary function which checks whether `AValue` is in the range defined by `AMin` and `AMax`, borders included.

Errors: If the variants are of different types and they cannot be converted, an exception will be raised.

See also: VarEnsureRange (2207)

85.4.37 VarInvalidArgError

Synopsis: Raise an `EVariantInvalidArgError` error.

Declaration:
```
procedure VarInvalidArgError
procedure VarInvalidArgError(AType: tvartype)
```

Visibility: default

Description: `VarInvalidArgError` raises an EVariantInvalidArgError (2218) exception with a standard error message. If the `AType` argument is specified, a description of the argument type is included in the error message.

See also: EVariantInvalidArgError (2218), VarInvalidNullOp (2208), VarInvalidOp (2208), VarBadIndexError (2205), VarArrayLockedError (2203)

85.4.38 VarInvalidNullOp

Synopsis: Raise an `EVariantInvalidOpError` error.

Declaration:
```
procedure VarInvalidNullOp
```

Visibility: default

Description: `VarInvalidNullOp` raises an EVariantInvalidOpError (2218) exception including null type description.

See also: EVariantInvalidOpError (2218), VarInvalidOp (2208)

85.4.39 VarInvalidOp

Synopsis: Raise a `EVariantInvalidOpError` error.

Declaration:
```
procedure VarInvalidOp
procedure VarInvalidOp(const aLeft: tvartype; const aRight: tvartype;
                       aOpCode: tvarop)
procedure VarInvalidOp(const aRight: tvartype; aOpCode: tvarop)
```

Visibility: default

Description: `VarInvalidOp` raises an EVariantInvalidOpError (2218) exception. when the left and/or right operand types (`aLeft`, `aRight` and the operation are specified, a description of the operand types and operation is included in the error message.

See also: EVariantInvalidOpError (2218), VarCastError (2206)

85.4.40 VarIsArray

Synopsis: Check whether a variant is an array.

Declaration:
```
function VarIsArray(const A: Variant) : Boolean
function VarIsArray(const A: Variant; AResolveByRef: Boolean) : Boolean
```

Visibility: default

Description: `VarIsArray` returns `True` if `A` is an array. If `AResolveByRef` is `True` (the default) then it will resolve all references first.

Errors: None.

See also: VarTypeIsValidArrayType (2216)

85.4.41 VarIsBool

Synopsis: Check if the variant is a boolean.

Declaration:
```
function VarIsBool(const V: Variant) : Boolean
```

Visibility: default

Description: `VarIsCustom` returns `True` if the variant `V` is a boolean value.

See also: VarIsEmpty (2210), VarIsNull (2211), VarIsCustom (2210), VarIsOrdinal (2211), VarIsFloat (2211), VarIsStr (2212)

85.4.42 VarIsByRef

Synopsis: Check if the variant is a reference to a value.

Declaration:
```
function VarIsByRef(const V: Variant) : Boolean
```

Visibility: default

Description: `VarIsByRef` checks whether the variant `V` is a reference to a value instead of an actual value. It returns `True` if this is the case, `False` if not.

See also: VarType (2216), VarAsType (2205), VarIsType (2212), VarIsEmpty (2210), VarIsNull (2211)

85.4.43 VarIsClear

Synopsis: Check if a variant is clear.

Declaration:
```
function VarIsClear(const V: Variant) : Boolean
```

Visibility: default

Description: `VarIsClear` returns `True` if the variant `V` is empty, or it is a dispatch type with value Nil. For custom types, the decision is left to the custom type.

See also: VarIsEmpty (2210), VarIsNull (2211), VarIsNumeric (2211), VarIsStr (2212)

85.4.44 VarIsCustom

Synopsis: Check if the variant is a custom value.

Declaration:
```
function VarIsCustom(const V: Variant) : Boolean
```

Visibility: default

Description: `VarIsCustom` returns `True` if the variant `V` is a custom variant (type is larger than CFirstUserType (2193)).

See also: VarIsEmpty (2210), VarIsNull (2211), VarIsOrdinal (2211), VarIsNumeric (2211), VarIsStr (2212)

85.4.45 VarIsEmpty

Synopsis: Check if the variant is empty.

Declaration:
```
function VarIsEmpty(const V: Variant) : Boolean
```

Visibility: default

Description: `VarIsEmpty` checks whether the variant `V` is empty (i.e. the type is `varEmpty`).

See also: VarType (2216), VarAsType (2205), VarIsType (2212), VarIsEmpty (2210), VarCheckEmpty (2206), VarIsNull (2211)

85.4.46 VarIsEmptyParam

Synopsis: Check if the variant is an error value for `VAR_PARAMNOTFOUND`.

Declaration:
```
function VarIsEmptyParam(const V: Variant) : Boolean
```

Visibility: default

Description: `VarIsEmptyParam` returns `True` if the variant `V` is an error typed variant with value `VAR_PARAMNOTFOUND`, or `False` otherwise.

See also: VarIsError (2210), SetClearVarToEmptyParam (2200)

85.4.47 VarIsError

Synopsis: Check if the variant has type `varError`.

Declaration:
```
function VarIsError(const V: Variant; out AResult: HRESULT) : Boolean
function VarIsError(const V: Variant) : Boolean
```

Visibility: default

Description: `VarIsError` checks `V` and returns `True` if the type is `varError`.

See also: VarIsNull (2211), VarIsEmpty (2210), VarIsOrdinal (2211), VarIsEmptyParam (2210)

85.4.48 VarIsFloat

Synopsis: Check if the variant is a floating-point value.

Declaration: `function VarIsFloat(const V: Variant) : Boolean`

Visibility: default

Description: `VarIsCustom` returns `True` if the variant `V` is of one of the ordinal types: FloatVarTypes (2194) (`varSingle`,`varDouble`,`varCurrency`).

See also: VarIsEmpty (2210), VarIsNull (2211), VarIsCustom (2210), VarIsOrdinal (2211), VarIsNumeric (2211), VarIsStr (2212)

85.4.49 VarIsNull

Synopsis: Check if a variant is null.

Declaration: `function VarIsNull(const V: Variant) : Boolean`

Visibility: default

Description: `VarIsNull` returns `True` if the type of the variant `V` is `varNull`. No dereferencing is done.

See also: VarIsEmpty (2210), VarType (2216), VarIsNumeric (2211), VarIsStr (2212)

85.4.50 VarIsNumeric

Synopsis: Check if the variant is a numerical value.

Declaration: `function VarIsNumeric(const V: Variant) : Boolean`

Visibility: default

Description: `VarIsCustom` returns `True` if the variant `V` is of one of the ordinal or floating point types: FloatVarTypes (2194), OrdinalVarTypes (2194)

See also: VarIsEmpty (2210), VarIsNull (2211), VarIsCustom (2210), VarIsOrdinal (2211), VarIsFloat (2211), VarIsStr (2212)

85.4.51 VarIsOrdinal

Synopsis: Check if the variant is an ordinal value.

Declaration: `function VarIsOrdinal(const V: Variant) : Boolean`

Visibility: default

Description: `VarIsCustom` returns `True` if the variant `V` is of one of the ordinal types: OrdinalVarTypes (2194) (`varSmallInt`,`varInteger`, `varBoolean`,`varShortInt`,`varByte`,`varWord`, `varLongWord`,`varInt64`)

See also: VarIsEmpty (2210), VarIsNull (2211), VarIsCustom (2210), VarIsFloat (2211), VarIsNumeric (2211), VarIsStr (2212)

85.4.52 VarIsStr

Synopsis: Check if the variant is a string.

Declaration:
```
function VarIsStr(const V: Variant) : Boolean
```

Visibility: default

Description: `VarIsCustom` returns `True` if the variant `V` is of one of the string types: `varOleStr`, `varUString`, `varString`

See also: VarIsEmpty (2210), VarIsNull (2211), VarIsCustom (2210), VarIsOrdinal (2211), VarIsFloat (2211), VarIsBool (2209)

85.4.53 VarIsType

Synopsis: Check if a variant is of certain type.

Declaration:
```
function VarIsType(const V: Variant; aVarType: tvartype) : Boolean
                   ;  Overload
function VarIsType(const V: Variant; const AVarTypes: Array of tvartype)
                    : Boolean;  Overload
```

Visibility: default

Description: `VarIsType` will return `True` if the variant `V` is of type `aVarType` or is in `aVarTypes`.

See also: VarType (2216), VarAsType (2205), VarIsByRef (2209), VarIsNull (2211), VarIsEmpty (2210)

85.4.54 VarNotImplError

Synopsis: Raise an `EVariantNotImplError` error.

Declaration:
```
procedure VarNotImplError
```

Visibility: default

Description: `VarNotImplError` raises an EVariantNotImplError (2219) exception with a standard error message.

See also: EVariantNotImplError (2219), VarInvalidNullOp (2208), VarInvalidOp (2208), VarBadIndexError (2205), VarArrayLockedError (2203)

85.4.55 VarOutOfMemoryError

Synopsis: Raise an `EVariantOutOfMemoryError` error.

Declaration:
```
procedure VarOutOfMemoryError
```

Visibility: default

Description: `VarOutOfMemoryError` raises an EVariantOutOfMemoryError (2219) exception with a standard error message.

See also: EVariantOutOfMemoryError (2219), VarInvalidNullOp (2208), VarInvalidOp (2208), VarBadIndexError (2205), VarArrayLockedError (2203)

85.4.56 VarOverflowError

Synopsis: Raise an `EVariantOverflowError` error.

Declaration:
```
procedure VarOverflowError
procedure VarOverflowError(const ASourceType: tvartype;
                           const ADestType: tvartype)
```

Visibility: default

Description: `VarOverflowError` raises a EVariantOverflowError (2219) exception with a description of the `ASourceType` and `ADestType` types in the message.

See also: EVariantOverflowError (2219), VarInvalidNullOp (2208), VarInvalidOp (2208)

85.4.57 VarRangeCheckError

Synopsis: Raise an `EVariantOverflowError` error.

Declaration:
```
procedure VarRangeCheckError(const AType: tvartype)
procedure VarRangeCheckError(const ASourceType: tvartype;
                             const ADestType: tvartype)
```

Visibility: default

Description: `VarRangeCheckError` raises an EVariantOverflowError (2219) exception with a standard error message. A description of the type `AType` is included in the error message, similarly if a `ASourceType` and `aDestType` source and destination types are provided.

See also: EVariantOverflowError (2219), VarOverflowError (2213), VarInvalidNullOp (2208), VarInvalidOp (2208), VarBadIndexError (2205), VarArrayLockedError (2203)

85.4.58 VarResultCheck

Synopsis: Check the result of an operation and raise exception if not OK.

Declaration:
```
procedure VarResultCheck(AResult: HRESULT)
procedure VarResultCheck(AResult: HRESULT; ASourceType: tvartype;
                         ADestType: tvartype)
```

Visibility: default

Description: `VarResultCheck` checks the result `aResult`. If it difers from `VAR_OK` then an appropriate exception is raised based on the error code. If a source and destination type `ASourceType` and `aDestType` are specified they are included in the error message.

See also: VarOverflowError (2213), VarInvalidNullOp (2208), VarInvalidOp (2208), VarBadIndexError (2205), VarArrayLockedError (2203)

85.4.59 VarSameValue

Synopsis: Check if 2 variants are the same.

Declaration:
```
function VarSameValue(const A: Variant; const B: Variant) : Boolean
```

Visibility: default

Description: `varSameValue` checks whether `A` and `B` have the same value. Here empty variants equal empty variants, and null variants equal null variants. For all other cases, the actual values are compared.

Errors: If the variants are of different types and they cannot be converted, an exception will be raised.

85.4.60 VarSupports

Synopsis: Check if a variant supports an interface.

Declaration:
```
function VarSupports(const V: Variant; const IID: TGuid; out Intf)
                         : Boolean
function VarSupports(const V: Variant; const IID: TGuid) : Boolean
```

Visibility: default

Description: `VarSupports` checks if the variant `V` contains an interface (types `Unknown, varDispatch`) and the interface supports the specified interface `IID`. If it does, `True` is returned, `False` otherwise. If `Intf` is specified, and the variant supports the requested interface, the interface instance is returned in `Intf`.

See also: #rtl.sysutils.Supports (1798)

85.4.61 VarToDateTime

Synopsis: Convert a variant to datetime value.

Declaration:
```
function VarToDateTime(const V: Variant) : TDateTime
```

Visibility: default

Description: `VarToDateTime` attempts to convert the variant `V` to a `TDateTime` value.

Errors: If the value is `Null`, an exception is raised.

See also: VarToUnicodeStr (2215), VarToStrDef (2214), VarIsStr (2212), VarToWideStr (2215), VarToWideStrDef (2215), VarFromDateTime (2207)

85.4.62 VarToStr

Synopsis: Convert a variant to string value.

Declaration:
```
function VarToStr(const V: Variant) : string
```

Visibility: default

Description: `VarToStr` attempts to convert the variant `V` to a string. If the value is `Null`, the result is an empty string.

See also: VarToStrDef (2214), VarIsStr (2212), VarToWideStr (2215)

85.4.63 VarToStrDef

Synopsis: Convert a variant to string value, specifying a default.

Declaration:
```
function VarToStrDef(const V: Variant; const ADefault: string) : string
```

Visibility: default

Description: `VarToStr` attempts to convert the variant `V` to a string. If the value is `Null`, the result is the specified `aDefault` string.

See also: VarToStrDef (2214), VarIsStr (2212), VarToStr (2214)

85.4.64 VarToUnicodeStr

Synopsis: Convert a variant to string value.

Declaration:
```
function VarToUnicodeStr(const V: Variant) : UnicodeString
```

Visibility: default

Description: `VarToUnicodeStr` attempts to convert the variant `V` to a string. If the value is `Null`, the result is an empty string.

See also: VarToWideStr (2215), VarToUnicodeStrDef (2215), VarIsStr (2212), VarToStr (2214), VarToUnicodeStr (2215)

85.4.65 VarToUnicodeStrDef

Synopsis: Convert a variant to string value, specifying a default.

Declaration:
```
function VarToUnicodeStrDef(const V: Variant;
                            const ADefault: UnicodeString)
                             : UnicodeString
```

Visibility: default

Description: `VarToUnicodeStr` attempts to convert the variant `V` to a string. If the value is `Null`, the result is the specified `aDefault` string.

See also: VarToUnicodeStr (2215), VarToStrDef (2214), VarIsStr (2212), VarToWideStr (2215), VarToWideStrDef (2215)

85.4.66 VarToWideStr

Synopsis: Convert a variant to string value.

Declaration:
```
function VarToWideStr(const V: Variant) : WideString
```

Visibility: default

Description: `VarToWideStr` attempts to convert the variant `V` to a string. If the value is `Null`, the result is an empty string.

See also: VarToWideStrDef (2215), VarIsStr (2212), VarToStr (2214), VarToUnicodeStr (2215)

85.4.67 VarToWideStrDef

Synopsis: Convert a variant to string value, specifying a default.

Declaration:
```
function VarToWideStrDef(const V: Variant; const ADefault: WideString)
                          : WideString
```

Visibility: default

Description: `VarToWideStr` attempts to convert the variant `V` to a string. If the value is `Null`, the result is the specified `aDefault` string.

See also: VarToWideStrDef (2215), VarIsStr (2212), VarToWideStr (2215), VarToUnicodeStrDef (2215)

85.4.68 VarType

Synopsis: Return the type of a variant.

Declaration:
```
function VarType(const V: Variant) : tvartype
```

Visibility: default

Description: `VarType` returns the type of the variant `V`. (it returns the internal type field of the variant structure)

See also: VarTypeDeRef (2216), VarIsType (2212), VarAsType (2205)

85.4.69 VarTypeAsText

Synopsis: Return a textual description of the variant type.

Declaration:
```
function VarTypeAsText(const AType: tvartype) : string
```

Visibility: default

Description: `VarTypeAsText` returns a textual description of the variant type `aType`. It is used in all exception raising routines to describe types, if a type is provided. For custom variant types, the name of the class is returned.

See also: VarOpAsText (2194)

85.4.70 VarTypeDeRef

Synopsis: Returns the resolved type of a variant.

Declaration:
```
function VarTypeDeRef(const V: Variant) : tvartype;  Overload
function VarTypeDeRef(const V: tvardata) : tvartype;  Overload
```

Visibility: default

Description: `VarTypeDeRef` will resolve the final type of the variant `V`: if `V` is a reference to another variant, the function will return the type of the referenced variant, recursing as much as needed till a non-variant type is encountered.

See also: VarType (2216), VarIsType (2212), VarAsType (2205)

85.4.71 VarTypeIsValidArrayType

Synopsis: Check if a variant type can be used in an array.

Declaration:
```
function VarTypeIsValidArrayType(const aVarType: tvartype) : Boolean
```

Visibility: default

Description: `VarTypeIsValidArrayType` checks if the variant type `aVarType` can be used as the type of a variant array. It returns `True` if it is usable, false otherwise. Currently, the following types are valid: SmallInt, Integer, Single, Double, Date, Currency, OleStr, Dispatch, Error, Boolean, Variant, Unknown (interface IUnknown), ShortInt, Byte, Word, LongWord.

See also: VarIsArray (2209), VarTypeIsValidElementType (2217)

85.4.72 VarTypeIsValidElementType

Synopsis: Check if a variant type can be used in an array.

Declaration: `function VarTypeIsValidElementType(const aVarType: tvartype) : Boolean`

Visibility: default

Description: `VarTypeIsValidElementType` checks if the variant type `aVarType` can be used as the type of a variant array, similar to VarTypeIsValidArrayType (2216). It returns `True` if it is usable, `False` otherwise. Currently, the following types are valid: SmallInt, Integer, Single, Double, Date, Currency, OleStr, Dispatch, Error, Boolean, Variant, Unknown (interface IUnknown), ShortInt, Byte, Word, LongWord int64. In addition, if the variant type is an known custom variant type, `True` is also returned.

Reference and array indicators are stripped before the test.

See also: VarIsArray (2209), VarTypeIsValidArrayType (2216)

85.4.73 VarUnexpectedError

Synopsis: Raise an `EVariantUnexpectedError` error.

Declaration: `procedure VarUnexpectedError`

Visibility: default

Description: `VarUnexpectedError` raises an EVariantUnexpectedError (2220) exception with a standard error message.

See also: EVariantUnexpectedError (2220), VarInvalidNullOp (2208), VarInvalidOp (2208), VarBadIndexError (2205), VarArrayLockedError (2203)

85.5 EVariantArrayCreateError

85.5.1 Description

`EVariantArrayCreateError` is the exception raised when a problem is detected during creation of an array. It can be raised manually by calling VarArrayCreateError (2202).

See also: VarArrayCreateError (2202)

85.6 EVariantArrayLockedError

85.6.1 Description

`EVariantArrayLockedError` is the exception raised when the `VAR_ARRAYISLOCKED` error is encountered in VarResultCheck (2213)

See also: VarResultCheck (2213)

85.7 EVariantBadIndexError

85.7.1 Description

`EVariantBadIndexError` is the exception raised when the `VAR_BADINDEX` error is encountered in VarResultCheck (2213)

See also: VarResultCheck (2213)

85.8 EVariantBadVarTypeError

85.8.1 Description

`EVariantBadVarTypeError` is the exception raised when the `VAR_BADVARTYPE` error is encountered in VarResultCheck (2213)

See also: VarResultCheck (2213)

85.9 EVariantDispatchError

85.9.1 Description

`EVariantDispatchError` is the exception raised when a dispatch call fails.

See also: TCustomVariantType (2222)

85.10 EVariantInvalidArgError

85.10.1 Description

`EVariantInvalidArgError` is the exception raised when the `VAR_INVALIDARG` error is encountered in VarResultCheck (2213)

See also: VarResultCheck (2213)

85.11 EVariantInvalidNullOpError

85.11.1 Description

`EVariantInvalidNullOpError` is defined for Delphi compatibility, but is not used in the FPC run time.

85.12 EVariantInvalidOpError

85.12.1 Description

`EVariantInvalidOpError` is the exception raised when the `VAR_EXCEPTION` error is encountered in VarResultCheck (2213)

See also: VarResultCheck (2213)

85.13 EVariantNotAnArrayError

85.13.1 Description

`EVariantNotAnArrayError` is not used in FPC and is defined for Delphi compatibility.

See also: VarResultCheck (2213)

85.14 EVariantNotImplError

85.14.1 Description

`EVariantNotImplError` is the exception raised when the `VAR_NOTIMPL` error is encountered in VarResultCheck (2213)

See also: VarResultCheck (2213)

85.15 EVariantOutOfMemoryError

85.15.1 Description

`EVariantOutOfMemoryError` is the exception raised when the `VAR_OUTOFMEMORY` error is encountered in VarResultCheck (2213) It can be raised manually by calling VarOutOfMemoryError (2212).

See also: VarOutOfMemoryError (2212), VarResultCheck (2213)

85.16 EVariantOverflowError

85.16.1 Description

`EVariantOverflowError` is the exception raised when the `VAR_OVERFLOW` error is encountered in VarResultCheck (2213)

See also: VarResultCheck (2213)

85.17 EVariantParamNotFoundError

85.17.1 Description

`EVariantParamNotFoundError` is the exception raised when the `VAR_PARAMNOTFOUND` error is encountered in VarResultCheck (2213)

See also: VarResultCheck (2213)

85.18 EVariantRangeCheckError

85.18.1 Description

`EVariantRangeCheckError` is defined for Delphi compatibility, it is not used in FPC.

85.19 EVariantTypeCastError

85.19.1 Description

`EVariantTypeCastError` is the exception raised when the `VAR_TYPEMISMATCH` error is encountered in VarResultCheck (2213)

See also: VarResultCheck (2213)

85.20 EVariantUnexpectedError

85.20.1 Description

`EVariantUnexpectedError` is the exception raised when the `VAR_UNEXPECTED` error is encountered in VarResultCheck (2213) It can be raised manually by calling VarUnexpectedError (2217).

See also: VarVarUnexpectedError (2193), VarResultCheck (2213)

85.21 IVarInstanceReference

85.21.1 Description

`IVarInstanceReference` is used to get the instance of an object from a custom variant. It has only one method, GetInstance (2220) which is used to retrieve the instance of a variant that contains an object.

The TPublishableVariantType (2229) descendent of TInvokeableVariantType (2227) uses this interface to implement reading published properties from the instance.

See also: IVarInstanceReference.GetInstance (2220)

85.21.2 Method overview

Page	Method	Description
2220	GetInstance	Return the object instance referenced by the custom variant.

85.21.3 IVarInstanceReference.GetInstance

Synopsis: Return the object instance referenced by the custom variant.

Declaration: `function GetInstance(const V: tvardata) : TObject`

Visibility: default

Description: `GetInstance` must return the object instance referenced by the custom variant `V`.

See also: TPublishableVariantType (2229)

85.22 IVarInvokeable

85.22.1 Description

`IVarInvokeable` must be implemented by the TCustomVariantType (2222) descendent if the custom variant needs to implement dynamic properties and methods.

`IVarInvokeable` has four functions that must be implemented. Any of these functions can be called whenever a custom variant's method is invoked or a property of the variant is read or written using `DispInvoke`.

The `TCustomVariantType` descendent TInvokeableVariantType (2227) implements the needed DispInvoke to call the `IVarInvokeable` interface, so you can descend from that type instead when creating a new custom variant type and override the needed functions.

See also: TInvokeableVariantType (2227), TCustomVariantType (2222)

85.22.2 Method overview

Page	Method	Description
2221	DoFunction	Called for methods that return a result (functions).
2221	DoProcedure	Called for methods that do not return a result (procedures).
2222	GetProperty	Called when a property must be read.
2222	SetProperty	Called when a property must be set.

85.22.3 IVarInvokeable.DoFunction

Synopsis: Called for methods that return a result (functions).

Declaration:
```
function DoFunction(var Dest: tvardata; const V: tvardata;
                    const Name: string; const Arguments: TVarDataArray)
                     : Boolean
```

Visibility: default

Description: `DoFunction` is called whenever a variant method that returns a result needs to be invoked. The `Dest` parameter points to the location where the result of the function must be placed; the result must be a variant value.

`V` is the variant on which the method is being executed. The `Name` is the name of the function to execute, and `Arguments` is an array of variant arguments that were passed to the function.

The function must return `True` if the function was executed correctly, `False` otherwise.

See also: IVarInvokeable.DoProcedure (2221), IVarInvokeable.GetProperty (2222), IVarInvokeable.SetProperty (2222)

85.22.4 IVarInvokeable.DoProcedure

Synopsis: Called for methods that do not return a result (procedures).

Declaration:
```
function DoProcedure(const V: tvardata; const Name: string;
                     const Arguments: TVarDataArray) : Boolean
```

Visibility: default

Description: `DoProcedure` is called whenever a variant method that does not return a result needs to be invoked. `V` is the variant on which the method is being executed. The `Name` is the name of the method to execute, and `Arguments` is an array of variant arguments that were passed to the method.

The function must return `True` if the procedure was executed correctly, `False` otherwise.

See also: IVarInvokeable.DoFunction (2221), IVarInvokeable.GetProperty (2222), IVarInvokeable.SetProperty (2222)

85.22.5 IVarInvokeable.GetProperty

Synopsis: Called when a property must be read.

Declaration:
```
function GetProperty(var Dest: tvardata; const V: tvardata;
                     const Name: string) : Boolean
```

Visibility: default

Description: `DoFunction` is called whenever a variant property is read. `Dest` must be filled with the value of the property on success. `V` is the variant on which the property is read. The `Name` is the name of the property to read.

The function must return `True` if the property was read correctly, `False` otherwise.

See also: IVarInvokeable.DoFunction (2221), IVarInvokeable.DoProcedure (2221), IVarInvokeable.SetProperty (2222)

85.22.6 IVarInvokeable.SetProperty

Synopsis: Called when a property must be set.

Declaration:
```
function SetProperty(var V: tvardata; const Name: string;
                     const Value: tvardata) : Boolean
```

Visibility: default

Description: `DoFunction` is called whenever a variant property is written. `Value` is filled with the new value of the property. `V` is the variant on which the property is written. The `Name` is the name of the property to write.

The function must return `True` if the property was written correctly, `False` otherwise.

See also: IVarInvokeable.DoFunction (2221), IVarInvokeable.DoProcedure (2221), IVarInvokeable.GetProperty (2222)

85.23 TCustomVariantType

85.23.1 Description

`TCustomVariantType` is used as a base class to implement custom variants. To define a custom variant type, a descendent of `TCustomVariant` must be made, and the appropriate methods must be overridden and implemented according to the specifications of the new type. Typically this means defining how your new type maps to another variant or a basic type.

Note that the `TCustomVariantType` descendent does not hold the data of the variant: it just describes how a variant record (TVarRec (1439)) that contains the new type's data can be examined or manipulated.

See also: #rtl.system.TVarRec (1439)

85.23.2 Interfaces overview

Page	Interfaces	Description
1399	IInterface	Basic interface for all COM based interfaces.

85.23.3 Method overview

Page	Method	Description
2225	BinaryOp	Perform a binary mathematical operation.
2224	Cast	Cast a custom variant to another type.
2224	CastTo	Cast a custom variant to another type.
2224	CastToOle	Cast variant value to OLE value.
2225	Clear	Clear a value.
2226	Compare	Comparison 2 custom variant values.
2226	CompareOp	Check result of a comparison between 2 custom variant values.
2225	Copy	Copy a custom variant value.
2223	Create	Instantiate a new custom variant type.
2223	Destroy	Unregister variant type.
2224	IsClear	Is the custom value unassigned ?
2226	UnaryOp	Perform a unary mathematical operation.

85.23.4 Property overview

Page	Properties	Access	Description
2227	VarType	r	Registered type.

85.23.5 TCustomVariantType.Create

Synopsis: Instantiate a new custom variant type.

Declaration:
```
constructor Create;  Overload
constructor Create(RequestedVarType: tvartype);  Overload
```

Visibility: public

Description: `Create` sets up the necessary reference counting mechanisms to act as an interface, and assigns the new variant type identifier. If specified, the `RequestedVarType` is the variant type that is registered. if it is zero, a new identifier is assigned.

See also: TCustomVariantType.Destroy (2223)

85.23.6 TCustomVariantType.Destroy

Synopsis: Unregister variant type.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` unregisters the custom variant type and removes the instance from memory. After this is called, variants with the (now unregistered) type can no longer be used.

See also: TCustomVariantType.Create (2223)

85.23.7 TCustomVariantType.IsClear

Synopsis: Is the custom value unassigned ?

Declaration:
```
function IsClear(const V: tvardata) : Boolean;  Virtual
```

Visibility: public

Description: `IsClear` must be overridden to decide whether the custom variant `V` is equivalent to `varClear` (unassigned) value. The VarIsClear (2209) function calls this method to decide whether a custom variant is unassigned.

The `TCustomVariantType` implementation of `IsClear` always returns `False`.

See also: VarIsClear (2209)

85.23.8 TCustomVariantType.Cast

Synopsis: Cast a custom variant to another type.

Declaration:
```
procedure Cast(var Dest: tvardata; const Source: tvardata);  Virtual
```

Visibility: public

Description: `Cast` is called when a variant value `Source` is cast to a custom variant `Dest`. Descendents must override this method to provide the desired conversion behaviour.

The `TCustomVariantType` implementation of `Cast` calls `CastTo` on the source variant with as destination type the custom variant type.

Errors: If the variant type cannot be cast, an exception is raised.

See also: TCustomVariantType.CastTo (2224), TCustomVariantType.CastToOle (2224)

85.23.9 TCustomVariantType.CastTo

Synopsis: Cast a custom variant to another type.

Declaration:
```
procedure CastTo(var Dest: tvardata; const Source: tvardata;
                 const aVarType: tvartype);  Virtual
```

Visibility: public

Description: `CastTo` is called when a custom variant value `Source` is cast to a variant type `aVarType` and stores the result in `Dest`. Descendents must override this method to provide the desired conversion behaviour.

See also: TCustomVariantType.Cast (2224), TCustomVariantType.CastToOle (2224)

85.23.10 TCustomVariantType.CastToOle

Synopsis: Cast variant value to OLE value.

Declaration:
```
procedure CastToOle(var Dest: tvardata; const Source: tvardata)
                    ;  Virtual
```

Visibility: public

Description: `CastToOle` is called when a variant must be cast to a value that can be used in OLE variants. Descendents must override this method to provide the desired conversion behaviour.

The `TCustomVariantType` implementation of `CastToOle` raises an 'unsupported' exception.

See also: TCustomVariantType.Cast (2224), TCustomVariantType.CastTo (2224)

85.23.11 TCustomVariantType.Clear

Synopsis: Clear a value.

Declaration:
```
procedure Clear(var V: tvardata);  Virtual;  Abstract
```

Visibility: public

Description: `Clear` is called when a variant value must be cleared, by `VarClear`. This is an abstract method that must be overridden and implemented in descendent classes. When the method returns, the type of the variant record must be `varClear`

If the custom variant value used some heap memory (for intance in an object) then this is the place to release that memory.

Errors: If this method is not overridden, an abstract error will be thrown.

See also: TCustomVariantType.IsClear (2224), TCustomVariantType.Copy (2225), VarClear (2206)

85.23.12 TCustomVariantType.Copy

Synopsis: Copy a custom variant value.

Declaration:
```
procedure Copy(var Dest: tvardata; const Source: tvardata;
                    const Indirect: Boolean);  Virtual;  Abstract
```

Visibility: public

Description: `Copy` is used when a custom variant value is assigned to another variant. This is an abstract method that must be overridden and implemented in descendent classes. When the method returns, the type of the `Dest` variant record must be equal to the `Source` type (the custom varant type value).

If `Indirect` is `True` source contains a reference to the custom variant. In that case, dest should also contain a reference.

Errors: If this method is not overridden, an abstract error will be thrown.

See also: TCustomVariantType.Clear (2225)

85.23.13 TCustomVariantType.BinaryOp

Synopsis: Perform a binary mathematical operation.

Declaration:
```
procedure BinaryOp(var Left: tvardata; const Right: tvardata;
                       const Operation: tvarop);  Virtual
```

Visibility: public

Description: `BinaryOp` performs `Operation` on `Left` and `Right` and stores the result of the result in `Left`. The `TCustomVariantType` implementation of this method raises an invalid operation error. Descendent classes must override this.

Errors: If a descendent does not override this, an invalid operation error is raised.

See also: =TCustomVariantType.UnaryOp (2193), =TCustomVariantType.CompareOp (2193)

85.23.14 TCustomVariantType.UnaryOp

Synopsis: Perform a unary mathematical operation.

Declaration:
```
procedure UnaryOp(var Right: tvardata; const Operation: tvarop)
                  ;  Virtual
```

Visibility: public

Description: `BinaryOp` performs `Operation` on `Right` and stores the result of the result in `Right`. The `TCustomVariantType` implementation of this method raises an invalid operation error. Descendent classes must override this method.

Errors: If a descendent does not override this, an invalid operation error is raised.

See also: =TCustomVariantType.BinaryOp (2193), =TCustomVariantType.CompareOp (2193)

85.23.15 TCustomVariantType.CompareOp

Synopsis: Check result of a comparison between 2 custom variant values.

Declaration:
```
function CompareOp(const Left: tvardata; const Right: tvardata;
                   const Operation: tvarop) : Boolean;  Virtual
```

Visibility: public

Description: `CompareOp` performs the comparison specified in `Operation` on `left` and `Right` and returns `True` if the values satisfy the requested operation. The `TCustomVariantType` implementation of this method raises an 'unsupported' error. Descendent classes must override this method.

Errors: If a descendent does not override this, an 'unsupported' error is raised.

See also: TCustomVariantType.Compare (2226), TCustomVariantType.UnaryOp (2226), TCustomVariantType.binaryOp (2225)

85.23.16 TCustomVariantType.Compare

Synopsis: Comparison 2 custom variant values.

Declaration:
```
procedure Compare(const Left: tvardata; const Right: tvardata;
                  var Relationship: TVarCompareResult);  Virtual
```

Visibility: public

Description: `Compare` performs the comparison specified in `Operation` on `left` and `Right` and returns the result of the comparison. The `TCustomVariantType` implementation of this method raises an 'unsupported' error. Descendent classes must override this method.

Errors: If a descendent does not override this, an 'unsupported' error is raised.

See also: TCustomVariantType.CompareOp (2226), TCustomVariantType.UnaryOp (2226), TCustomVariantType.binaryOp (2225)

85.23.17 TCustomVariantType.VarType

Synopsis: Registered type.

Declaration: `Property VarType : tvartype`

Visibility: public

Access: Read

Description: `VarType` is the custom variant type identifier for the custom variant values. It is assigned by the system in the consutructor.

See also: TCustomVariantType.Create (2223)

85.24 TInvokeableVariantType

85.24.1 Description

`TInvokeableVariantType` is a TCustomVariantType (2222) descendent which implements the TCustomVariantType.DispInvoke (2222) method. It translates the `DispInvoke"` method to calls to one or more of the four methods of IVarInvokeable (2221): DoFunction (2221) or DoProcedure (2221) for method calls (DISPATCH_METHOD) and GetProperty (2222) for property reading (DISPATCH_PROPERTYGET) or SetProperty (2222) for property writing (DISPATCH_PROPERTYSET). Other combinations are possible. It has empty stubs for these methods, which must be implemented in a descendent that is used to describe a custom variant.

See also: IVarInvokeable.DoFunction (2221), IVarInvokeable.DoProcedure (2221), IVarInvokeable.GetProperty (2222), IVarInvokeable.SetProperty (2222)

85.24.2 Interfaces overview

Page	Interfaces	Description
2221	IVarInvokeable	Interface needed for custom variants that need to implement dynamic properties and methods.

85.24.3 Method overview

Page	Method	Description
2227	DoFunction	Empty stub for `IVarInvokeable.DoFunction`, to be implemented in descendents.
2228	DoProcedure	Empty stub for `IVarInvokeable.DoProcedure`, to be implemented in descendents.
2228	GetProperty	Empty stub for `IVarInvokeable.GetProperty`, to be implemented in descendents.
2228	SetProperty	Empty stub for `IVarInvokeable.SetProperty`, to be implemented in descendents.

85.24.4 TInvokeableVariantType.DoFunction

Synopsis: Empty stub for `IVarInvokeable.DoFunction`, to be implemented in descendents.

Declaration:
```
function DoFunction(var Dest: tvardata; const V: tvardata;
                    const Name: string; const Arguments: TVarDataArray)
                     : Boolean;  Virtual
```

Visibility: public

Description: `DoFunction` is the default implementation of IVarInvokeable.DoFunction (2221), which always returns `False`. It must be overridden in descendent classes to implement actual calling of a function.

See also: IVarInvokeable.DoFunction (2221)

85.24.5 TInvokeableVariantType.DoProcedure

Synopsis: Empty stub for `IVarInvokeable.DoProcedure`, to be implemented in descendents.

Declaration:
```
function DoProcedure(const V: tvardata; const Name: string;
                     const Arguments: TVarDataArray) : Boolean;  Virtual
```

Visibility: public

Description: `DoProcedure` is the default implementation of IVarInvokeable.DoProcedure (2221), which always returns `False`. It must be overridden in descendent classes to implement actual calling of a procedure.

See also: IVarInvokeable.DoProcedure (2221)

85.24.6 TInvokeableVariantType.GetProperty

Synopsis: Empty stub for `IVarInvokeable.GetProperty`, to be implemented in descendents.

Declaration:
```
function GetProperty(var Dest: tvardata; const V: tvardata;
                     const Name: string) : Boolean;  Virtual
```

Visibility: public

Description: `GetProperty` is the default implementation of IVarInvokeable.GetProperty (2222), which always returns `False`. It must be overridden in descendent classes to implement actual reading of a property.

See also: IVarInvokeable.GetProperty (2222)

85.24.7 TInvokeableVariantType.SetProperty

Synopsis: Empty stub for `IVarInvokeable.SetProperty`, to be implemented in descendents.

Declaration:
```
function SetProperty(var V: tvardata; const Name: string;
                     const Value: tvardata) : Boolean;  Virtual
```

Visibility: public

Description: `SetProperty` is the default implementation of IVarInvokeable.SetProperty (2222), which always returns `False`. It must be overridden in descendent classes to implement actual writing of a property.

See also: IVarInvokeable.SetProperty (2222)

85.25 TPublishableVariantType

85.25.1 Description

`TPublishableVariantType` implements the IVarInvokeable.GetProperty (2222) and IVarInvokeable.SetProperty (2222) methods by getting or setting the published properties of the class instance returned by the `IVarInstanceReference` interface it implements.

See also: TInvokeableVariantType (2227), IVarInvokeable.GetProperty (2222), IVarInvokeable.SetProperty (2222), IVarInstanceReference (2220)

85.25.2 Interfaces overview

Page	Interfaces	Description
2220	IVarInstanceReference	Interface for variants that refer to an object.

85.25.3 Method overview

Page	Method	Description
2229	GetProperty	Read the property.
2229	SetProperty	Write the property.

85.25.4 TPublishableVariantType.GetProperty

Synopsis: Read the property.

Declaration:
```
function GetProperty(var Dest: tvardata; const V: tvardata;
                     const Name: string) : Boolean;  Override
```

Visibility: public

Description: `GetProperty` implements reading the property by looking for the property in the published properties of the instance returned by the IVarInstanceReference (2220) interface, and returning the value as a variant.

See also: IVarInstanceReference (2220), IVarInvokeable.GetProperty (2222)

85.25.5 TPublishableVariantType.SetProperty

Synopsis: Write the property.

Declaration:
```
function SetProperty(var V: tvardata; const Name: string;
                     const Value: tvardata) : Boolean;  Override
```

Visibility: public

Description: `SetProperty` implements writing the property by looking for the property in the published properties of the instance returned by the IVarInstanceReference (2220) interface, and setting the value as a variant.

See also: IVarInstanceReference (2220), IVarInvokeable.SetProperty (2222)

Chapter 86

Reference for unit 'video'

86.1 Used units

Table 86.1: Used units by unit 'video'

Name	Page
System	1362

86.2 Overview

The Video unit implements a screen access layer which is system independent. It can be used to write on the screen in a system-independent way, which should be optimal on all platforms for which the unit is implemented.

The working of the Video is simple: After calling InitVideo (2247), the array `VideoBuf` contains a representation of the video screen of size `ScreenWidth*ScreenHeight`, going from left to right and top to bottom when walking the array elements: `VideoBuf[0]` contains the character and color code of the top-left character on the screen. `VideoBuf[ScreenWidth]` contains the data for the character in the first column of the second row on the screen, and so on.

To write to the 'screen', the text to be written should be written to the `VideoBuf` array. Calling UpdateScreen (2251) will then copy the text to the screen in the most optimal way. (an example can be found further on).

The color attribute is a combination of the foreground and background color, plus the blink bit. The bits describe the various color combinations:

bits 0-3 The foreground color. Can be set using all color constants.

bits 4-6 The background color. Can be set using a subset of the color constants.

bit 7 The blinking bit. If this bit is set, the character will appear blinking.

Each possible color has a constant associated with it, see the constants section for a list of constants.

The foreground and background color can be combined to a color attribute with the following code:

```
Attr:=ForeGroundColor + (BackGroundColor shl 4);
```

The color attribute can be logically or-ed with the blink attribute to produce a blinking character:

```
Atrr:=Attr or blink;
```

But not all drivers may support this.

The contents of the `VideoBuf` array may be modified: This is 'writing' to the screen. As soon as everything that needs to be written in the array is in the `VideoBuf` array, calling `UpdateScreen` will copy the contents of the array screen to the screen, in a manner that is as efficient as possible.

The updating of the screen can be prohibited to optimize performance; To this end, the LockScreenUpdate (2248) function can be used: This will increment an internal counter. As long as the counter differs from zero, calling UpdateScreen (2251) will not do anything. The counter can be lowered with UnlockScreenUpdate (2250). When it reaches zero, the next call to UpdateScreen (2251) will actually update the screen. This is useful when having nested procedures that do a lot of screen writing.

The video unit also presents an interface for custom screen drivers, thus it is possible to override the default screen driver with a custom screen driver, see the SetVideoDriver (2250) call. The current video driver can be retrieved using the GetVideoDriver (2245) call.

Remark The video unit should *not* be used together with the crt unit. Doing so will result in very strange behaviour, possibly program crashes.

86.3 Examples utility unit.

The examples in this section make use of the unit vidutil, which contains the `TextOut` function. This function writes a text to the screen at a given location. It looks as follows:

Listing: ./examples/videoex/vidutil.pp

```
unit vidutil;

Interface

uses
  video;

Procedure TextOut(X,Y : Word;Const S : String);

Implementation

Procedure TextOut(X,Y : Word;Const S : String);

Var
  W,P,I,M : Word;

begin
  P:=((X-1)+(Y-1)*ScreenWidth);
  M:=Length(S);
  If P+M>ScreenWidth*ScreenHeight then
    M:=ScreenWidth*ScreenHeight-P;
  For I:=1 to M do
    VideoBuf^[P+I-1]:=Ord(S[i])+($07 shl 8);
end;

end.
```

86.4 Writing a custom video driver.

Writing a custom video driver is not difficult, and generally means implementing a couple of functions, which whould be registered with the SetVideoDriver (2250) function. The various functions that can be implemented are located in the TVideoDriver (2252) record:

```
TVideoDriver = Record
  InitDriver        : Procedure;
  DoneDriver        : Procedure;
  UpdateScreen      : Procedure(Force : Boolean);
  ClearScreen       : Procedure;
  SetVideoMode      : Function (Const Mode : TVideoMode) : Boolean;
  GetVideoModeCount : Function : Word;
  GetVideoModeData  : Function(Index : Word; Var Data : TVideoMode) : Boolean;
  SetCursorPos      : procedure (NewCursorX, NewCursorY: Word);
  GetCursorType     : function : Word;
  SetCursorType     : procedure (NewType: Word);
  GetCapabilities   : Function : Word;
end;
```

Not all of these functions must be implemented. In fact, the only absolutely necessary function to write a functioning driver is the `UpdateScreen` function. The general calls in the Video unit will check which functionality is implemented by the driver.

The functionality of these calls is the same as the functionality of the calls in the video unit, so the expected behaviour can be found in the previous section. Some of the calls, however, need some additional remarks.

InitDriver Called by `InitVideo`, this function should initialize any data structures needed for the functionality of the driver, maybe do some screen initializations. The function is guaranteed to be called only once; It can only be called again after a call to `DoneVideo`. The variables `ScreenWidth` and `ScreenHeight` should be initialized correctly after a call to this function, as the `InitVideo` call will initialize the `VideoBuf` and `OldVideoBuf` arrays based on their values.

DoneDriver This should clean up any structures that have been initialized in the `InitDriver` function. It should possibly also restore the screen as it was before the driver was initialized. The VideoBuf and `OldVideoBuf` arrays will be disposed of by the general `DoneVideo` call.

UpdateScreen This is the only required function of the driver. It should update the screen based on the `VideoBuf` array's contents. It can optimize this process by comparing the values with values in the `OldVideoBuf` array. After updating the screen, the `UpdateScreen` procedure should update the `OldVideoBuf` by itself. If the `Force` parameter is `True`, the whole screen should be updated, not just the changed values.

ClearScreen If there is a faster way to clear the screen than to write spaces in all character cells, then it can be implemented here. If the driver does not implement this function, then the general routines will write spaces in all video cells, and will call `UpdateScreen(True)`.

SetVideoMode Should set the desired video mode, if available. It should return `True` if the mode was set, `False` if not.

GetVideoModeCount Should return the number of supported video modes. If no modes are supported, this function should not be implemented; the general routines will return 1. (for the current mode)

GetVideoModeData Should return the data for the `Index`-th mode; `Index` is zero based. The function should return true if the data was returned correctly, false if `Index` contains an invalid index. If this is not implemented, then the general routine will return the current video mode when `Index` equals 0.

GetCapabilities If this function is not implemented, zero (i.e. no capabilities) will be returned by the general function.

The following unit shows how to override a video driver, with a driver that writes debug information to a file. The unit can be used in any of the demonstration programs, by simply including it in the `uses` clause. Setting `DetailedVideoLogging` to `True` will create a more detailed log (but will also slow down functioning)

Listing: ./examples/videoex/viddbg.pp

```
unit viddbg;

Interface

uses video;


Procedure StartVideoLogging;
Procedure StopVideoLogging;
Function  IsVideoLogging : Boolean;
Procedure  SetVideoLogFileName(FileName : String);

Const
  DetailedVideoLogging : Boolean = False;

Implementation

uses sysutils,keyboard;

var
  NewVideoDriver,
  OldVideoDriver : TVideoDriver;
  Active,Logging : Boolean;
  LogFileName : String;
  VideoLog : Text;

Function TimeStamp : String;

begin
  TimeStamp:=FormatDateTime('hh:nn:ss',Time());
end;

Procedure StartVideoLogging;

begin
  Logging:=True;
  Writeln(VideoLog,'Start logging video operations at: ',TimeStamp);
end;

Procedure StopVideoLogging;

begin
  Writeln(VideoLog,'Stop logging video operations at: ',TimeStamp);
  Logging:=False;
```

```
end;

Function IsVideoLogging : Boolean;

begin
  IsVideoLogging:=Logging;
end;

Var
  ColUpd,RowUpd : Array[0..1024] of Integer;

Procedure DumpScreenStatistics(Force : Boolean);

Var
  I,Count : Integer;

begin
  If Force then
    Write(VideoLog,'forced ');
  Writeln(VideoLog,'video update at ',TimeStamp,' : ');
  FillChar(Colupd,SizeOf(ColUpd),#0);
  FillChar(Rowupd,SizeOf(RowUpd),#0);
  Count:=0;
  For I:=0 to VideoBufSize div SizeOf(TVideoCell) do
    begin
    If VideoBuf^[i]<>OldVideoBuf^[i] then
      begin
      Inc(Count);
      Inc(ColUpd[I mod ScreenWidth]);
      Inc(RowUpd[I div ScreenHeight]);
      end;
    end;
  Write(VideoLog,Count,' videocells differed divided over ');
  Count:=0;
  For I:=0 to ScreenWidth-1 do
    If ColUpd[I]<>0 then
      Inc(Count);
  Write(VideoLog,Count,' columns and ');
  Count:=0;
  For I:=0 to ScreenHeight-1 do
    If RowUpd[I]<>0 then
      Inc(Count);
  Writeln(VideoLog,Count,' rows.');
  If DetailedVideoLogging Then
   begin
   For I:=0 to ScreenWidth-1 do
     If (ColUpd[I]<>0) then
       Writeln(VideoLog,'Col ',i,' : ',ColUpd[I]:3,' rows changed');
   For I:=0 to ScreenHeight-1 do
     If (RowUpd[I]<>0) then
       Writeln(VideoLog,'Row ',i,' : ',RowUpd[I]:3,' colums changed');
   end;
end;

Procedure LogUpdateScreen(Force : Boolean);

begin
  If Logging then
```

```
    DumpScreenStatistics(Force);
  OldVideoDriver.UpdateScreen(Force);
end;

Procedure LogInitVideo;

begin
  OldVideoDriver.InitDriver();
  Assign(VideoLog,logFileName);
  Rewrite(VideoLog);
  Active:=True;
  StartVideoLogging;
end;

Procedure LogDoneVideo;

begin
  StopVideoLogging;
  Close(VideoLog);
  Active:=False;
  OldVideoDriver.DoneDriver();
end;

Procedure SetVideoLogFileName(FileName : String);

begin
  If Not Active then
    LogFileName:=FileName;
end;

Initialization
  GetVideoDriver(OldVideoDriver);
  NewVideoDriver:=OldVideoDriver;
  NewVideoDriver.UpdateScreen:=@LogUpdateScreen;
  NewVideoDriver.InitDriver:=@LogInitVideo;
  NewVideoDriver.DoneDriver:=@LogDoneVideo;
  LogFileName:='Video.log';
  Logging:=False;
  SetVideoDriver(NewVideoDriver);
end.
```

86.5 Constants, types and variables

86.5.1 Constants

```
Black = 0
```

Black color attribute.

```
Blink = 128
```

Blink attribute.

```
Blue = 1
```

Blue color attribute.

```
Brown = 6
```

Brown color attribute.

```
cpBlink = $0002
```

Video driver supports blink attribute.

```
cpChangeCursor = $0020
```

Video driver supports changing cursor shape.

```
cpChangeFont = $0008
```

Video driver supports changing screen font.

```
cpChangeMode = $0010
```

Video driver supports changing mode.

```
cpColor = $0004
```

Video driver supports color.

```
cpUnderLine = $0001
```

Video driver supports underline attribute.

```
crBlock = 2
```

Block cursor.

```
crHalfBlock = 3
```

Half block cursor.

```
crHidden = 0
```

Hide cursor.

```
crUnderLine = 1
```

Underline cursor.

```
Cyan = 3
```

Cyan color attribute.

```
DarkGray = 8
```

Dark gray color attribute.

```
errOk = 0
```

No error.

```
ErrorCode : LongInt = ErrOK
```

Error code returned by the last operation.

```
ErrorHandler : TErrorHandler = @ DefaultErrorHandler
```

The `ErrorHandler` variable can be set to a custom-error handling function. It is set by default to the DefaultErrorHandler (2242) function.

```
ErrorInfo : Pointer = Nil
```

Pointer to extended error information.

```
errVioBase = 1000
```

Base value for video errors.

```
errVioInit = errVioBase + 1
```

Video driver initialization error.

```
errVioNoSuchMode = errVioBase + 3
```

Invalid video mode.

```
errVioNotSupported = errVioBase + 2
```

Unsupported video function.

```
FVMaxWidth = 240
```

Maximum screen buffer width.

```
Green = 2
```

Green color attribute.

```
iso_codepages = [iso01, iso02, iso03, iso04, iso05, iso06, iso07,
  iso08, iso09, iso10, iso13, iso14, iso15]
```

`iso_codepages` is a set containing all code pages that use an ISO encoding.

```
LightBlue = 9
```

Light Blue color attribute.

```
LightCyan = 11
```

Light cyan color attribute.

```
LightGray = 7
```

Light gray color attribute.

```
LightGreen = 10
```

Light green color attribute.

```
LightMagenta = 13
```

Light magenta color attribute.

```
LightRed = 12
```

Light red color attribute.

```
LowAscii = True
```

On some systems, the low 32 values of the DOS code page are necessary for the ASCII control codes and cannot be displayed by programs. If LowAscii is true, you can use the low 32 ASCII values. If it is false, you must avoid using them.

LowAscii can be implemented either through a constant, variable or property. You should under no circumstances assume that you can write to LowAscii, or take its address.

```
Magenta = 5
```

Magenta color attribute.

```
NoExtendedFrame = False
```

The VT100 character set only has line drawing characters consisting of a single line. If this value is true, the line drawing characters with two lines will be automatically converted to single lines.

NoExtendedFrame can be implemented either through a constant, variable or property. You should under no circumstances assume that you can write to NoExtendedFrame, or take its address.

```
Red = 4
```

Red color attribute.

```
ScreenHeight : Word = 0
```

Current screen height.

```
ScreenWidth : Word = 0
```

Current screen Width.

```
vga_codepages = [cp437, cp850, cp852, cp866]
```

vga_codepages is a set containing all code pages that can be considered a normal vga font (as in use on early VGA cards) Note that KOI8-R has line drawing characters in wrong place.

```
vioOK = 0
```

No errors occurred.

```
White = 15
```

White color attribute.

```
Yellow = 14
```

Yellow color attribute.

86.5.2 Types

```
PVideoBuf = ^TVideoBuf
```

Pointer type to TVideoBuf (2240).

```
PVideoCell = ^TVideoCell
```

Pointer type to TVideoCell (2240).

```
PVideoMode = ^TVideoMode
```

Pointer to TVideoMode (2252) record.

```
TErrorHandler = function(Code: LongInt; Info: Pointer)
  : TErrorHandlerReturnValue
```

The TErrorHandler function is used to register an own error handling function. It should be used when installing a custom error handling function, and must return one of the above values.

Code should contain the error code for the error condition, and the Info parameter may contain any data type specific to the error code passed to the function.

```
TErrorHandlerReturnValue = (errRetry,errAbort,errContinue)
```

Table 86.2: Enumeration values for type TErrorHandlerReturnValue

Value	Explanation
errAbort	abort and return error code.
errContinue	abort without returning an errorcode.
errRetry	retry the operation.

Type used to report and respond to error conditions.

```
TVideoBuf = Array[0..32759] of TVideoCell
```

The `TVideoBuf` type represents the screen.

```
TVideoCell = Word
```

`TVideoCell` describes one character on the screen. One of the bytes contains the color attribute with which the character is drawn on the screen, and the other byte contains the ASCII code of the character to be drawn. The exact position of the different bytes in the record is operating system specific. On most little-endian systems, the high byte represents the color attribute, while the low-byte represents the ASCII code of the character to be drawn.

```
TVideoModeSelector = function(const VideoMode: TVideoMode;
  Params: LongInt) : Boolean
```

Video mode selection callback prototype.

86.5.3 Variables

```
CursorLines : Byte
```

`CursorLines` is a bitmask which determines which cursor lines are visible and which are not. Each set bit corresponds to a cursorline being shown.

This variable is not supported on all platforms, so it should be used sparingly.

```
CursorX : Word
```

Current horizontal position in the screen where items will be written.

```
CursorY : Word
```

Current vertical position in the screen where items will be written.

```
external_codepage : Tencoding
```

This variable is for internal use only and should not be used.

```
internal_codepage : Tencoding
```

This variable is for internal use only and should not be used.

```
OldVideoBuf : PVideoBuf
```

The `OldVideoBuf` contains the state of the video screen after the last screen update. The UpdateScreen (2251) function uses this array to decide which characters on screen should be updated, and which not.

Note that the `OldVideoBuf` array may be ignored by some drivers, so it should not be used. The Array is in the interface section of the video unit mainly so drivers that need it can make use of it.

```
ScreenColor : Boolean
```

`ScreenColor` indicates whether the current screen supports colors.

`VideoBuf : PVideoBuf`

`VideoBuf` forms the heart of the Video unit: This variable represents the physical screen. Writing to this array and calling UpdateScreen (2251) will write the actual characters to the screen.

`VideoBufSize : LongInt`

Current size of the video buffer pointed to by VideoBuf (2241)

86.6 Procedures and functions

86.6.1 ClearScreen

Synopsis: Clear the video screen.

Declaration: `procedure ClearScreen`

Visibility: default

Description: `ClearScreen` clears the entire screen, and calls UpdateScreen (2251) after that. This is done by writing spaces to all character cells of the video buffer in the default color (lightgray on black, color attribute \$07).

Errors: None.

See also: InitVideo (2247), UpdateScreen (2251)

Listing: ./examples/videoex/ex3.pp

```
program testvideo;

uses video,keyboard,vidutil;

Var
  i : longint;
  k : TkeyEvent;

begin
  InitVideo;
  InitKeyboard;
  For I:=1 to 10 do
    TextOut(i,i, 'Press any key to clear screen');
  UpdateScreen(false);
  K:=GetKeyEvent;
  ClearScreen;
  TextOut(1,1,'Cleared screen. Press any key to end');
  UpdateScreen(true);
  K:=GetKeyEvent;
  DoneKeyBoard;
  DoneVideo;
end.
```

86.6.2 DefaultErrorHandler

Synopsis: Default error handling routine.

Declaration:
```
function DefaultErrorHandler(AErrorCode: LongInt; AErrorInfo: Pointer)
                                  : TErrorHandlerReturnValue
```

Visibility: default

Description: `DefaultErrorHandler` is the default error handler used by the video driver. It simply sets the error code `AErrorCode` and `AErrorInfo` in the global variables `ErrorCode` and `ErrorInfo` and returns `errContinue`.

Errors: None.

86.6.3 DoneVideo

Synopsis: Disable video driver.

Declaration:
```
procedure DoneVideo
```

Visibility: default

Description: `DoneVideo` disables the Video driver if the video driver is active. If the videodriver was already disabled or not yet initialized, it does nothing. Disabling the driver means it will clean up any allocated resources, possibly restore the screen in the state it was before `InitVideo` was called. Particularly, the `VideoBuf` and `OldVideoBuf` arrays are no longer valid after a call to `DoneVideo`.

The `DoneVideo` should always be called if `InitVideo` was called. Failing to do so may leave the screen in an unusable state after the program exits.

For an example, see most other functions.

Errors: Normally none. If the driver reports an error, this is done through the `ErrorCode` variable.

See also: InitVideo (2247)

86.6.4 GetCapabilities

Synopsis: Get current driver capabilities.

Declaration:
```
function GetCapabilities : Word
```

Visibility: default

Description: `GetCapabilities` returns the capabilities of the current driver. It is an or-ed combination of the following constants:

cpUnderLine Video driver supports underline attribute.

cpBlink Video driver supports blink attribute.

cpColor Video driver supports color.

cpChangeFont Video driver supports changing screen font.

cpChangeMode Video driver supports changing mode.

cpChangeCursor Video driver supports changing cursor shape.

Note that the video driver should not yet be initialized to use this function. It is a property of the driver.

Errors: None.

See also: GetCursorType (2243), GetVideoDriver (2245)

Listing: ./examples/videoex/ex4.pp

```
Program Example4;

{ Program to demonstrate the GetCapabilities function. }

Uses video;

Var
  W: Word;

  Procedure TestCap(Cap: Word; Msg : String);

  begin
    Write(Msg,' : ');
    If (W and Cap=Cap) then
      Writeln('Yes')
    else
      Writeln('No');
  end;

begin
  W:=GetCapabilities;
  Writeln('Video driver supports following functionality');
  TestCap(cpUnderLine,'Underlined characters');
  TestCap(cpBlink,'Blinking characters');
  TestCap(cpColor,'Color characters');
  TestCap(cpChangeFont,'Changing font');
  TestCap(cpChangeMode,'Changing video mode');
  TestCap(cpChangeCursor,'Changing cursor shape');
end.
```

86.6.5 GetCursorType

Synopsis: Get screen cursor type.

Declaration: `function GetCursorType : Word`

Visibility: default

Description: `GetCursorType` returns the current cursor type. It is one of the following values:

crHiddenHide cursor.

crUnderLineUnderline cursor.

crBlockBlock cursor.

crHalfBlockHalf block cursor.

Note that not all drivers support all types of cursors.

Errors: None.

See also: SetCursorType (2249), GetCapabilities (2242)

Listing: ./examples/videoex/ex5.pp

```
Program Example5;

{ Program to demonstrate the GetCursorType function. }

Uses video,keyboard,vidutil;

Const
  Cursortypes : Array[crHidden..crHalfBlock] of string =
    ('Hidden','UnderLine','Block','HalfBlock');

begin
  InitVideo;
  InitKeyboard;
  TextOut(1,1,'Cursor type: '+CursorTypes[GetCursorType]);
  TextOut(1,2,'Press any key to exit.');
  UpdateScreen(False);
  GetKeyEvent;
  DoneKeyboard;
  DoneVideo;
end.
```

86.6.6 GetLockScreenCount

Synopsis: Get the screen lock update count.

Declaration: `function GetLockScreenCount : Integer`

Visibility: default

Description: `GetLockScreenCount` returns the current lock level. When the lock level is zero, a call to UpdateScreen (2251) will actually update the screen.

Errors: None.

See also: LockScreenUpdate (2248), UnlockScreenUpdate (2250), UpdateScreen (2251)

Listing: ./examples/videoex/ex6.pp

```
Program Example6;

{ Program to demonstrate the GetLockScreenCount function. }

Uses video,keyboard,vidutil;

Var
  I : Longint;
  S : String;

begin
  InitVideo;
  InitKeyboard;
  TextOut(1,1,'Press key till new text appears.');
  UpdateScreen(False);
  Randomize;
  For I:=0 to Random(10)+1 do
    LockScreenUpdate;
```

```
  I:=0;
  While GetLockScreenCount<>0 do
    begin
    Inc(I);
    Str(I,S);
    UnlockScreenUpdate;
    GetKeyEvent;
    TextOut(1,1,'UnLockScreenUpdate had to be called '+S+' times');
    UpdateScreen(False);
    end;
  TextOut(1,2,'Press any key to end.');
  UpdateScreen(False);
  GetKeyEvent;
  DoneKeyboard;
  DoneVideo;
end.
```

86.6.7 GetVideoDriver

Synopsis: Get a copy of the current video driver.

Declaration: `procedure GetVideoDriver(var Driver: TVideoDriver)`

Visibility: default

Description: `GetVideoDriver` returns the currently active video driver record in `Driver`. It can be used to clone the current video driver, or to override certain parts of it using the SetVideoDriver (2250) call.

Errors: None.

See also: SetVideoDriver (2250)

86.6.8 GetVideoMode

Synopsis: Return current video mode.

Declaration: `procedure GetVideoMode(var Mode: TVideoMode)`

Visibility: default

Description: `GetVideoMode` returns the settings of the currently active video mode. The `row,col` fields indicate the dimensions of the current video mode, and `Color` is true if the current video supports colors.

See also: SetVideoMode (2250), GetVideoModeData (2247)

Listing: ./examples/videoex/ex7.pp

```
Program Example7;

{ Program to demonstrate the GetVideoMode function. }

Uses video,keyboard,vidutil;

Var
  M : TVideoMode;
  S : String;
```

```
begin
  InitVideo;
  InitKeyboard;
  GetVideoMode(M);
  if M.Color then
    TextOut(1,1,'Current mode has color')
  else
    TextOut(1,1,'Current mode does not have color');
  Str(M.Row,S);
  TextOut(1,2,'Number of rows    : '+S);
  Str(M.Col,S);
  TextOut(1,3,'Number of columns : '+S);
  Textout(1,4,'Press any key to exit.');
  UpdateScreen(False);
  GetKeyEvent;
  DoneKeyboard;
  DoneVideo;
end.
```

86.6.9 GetVideoModeCount

Synopsis: Get the number of video modes supported by the driver.

Declaration: `function GetVideoModeCount : Word`

Visibility: default

Description: `GetVideoModeCount` returns the number of video modes that the current driver supports. If the driver does not support switching of modes, then 1 is returned.

This function can be used in conjunction with the GetVideoModeData (2247) function to retrieve data for the supported video modes.

Errors: None.

See also: GetVideoModeData (2247), GetVideoMode (2245)

Listing: ./examples/videoex/ex8.pp

```
Program Example8;

{ Program to demonstrate the GetVideoModeCount function. }

Uses video,keyboard,vidutil;

Procedure DumpMode (M : TVideoMode; Index : Integer);

Var
 S : String;

begin
  Str(Index:2,S);
  inc(Index);
  TextOut(1,Index,'Data for mode '+S+': ');
  if M.Color then
    TextOut(19,Index,'   color,')
  else
```

```
    TextOut(19,Index,'No color,');
  Str(M.Row:3,S);
  TextOut(28,Index,S+' rows');
  Str(M.Col:3,S);
  TextOut(36,index,S+' columns');
end;

Var
  i,Count : Integer;
  m : TVideoMode;

begin
  InitVideo;
  InitKeyboard;
  Count:=GetVideoModeCount;
  For I:=1 to Count do
    begin
    GetVideoModeData(I-1,M);
    DumpMode(M,I-1);
    end;
  TextOut(1,Count+1,'Press any key to exit');
  UpdateScreen(False);
  GetKeyEvent;
  DoneKeyboard;
  DoneVideo;
end.
```

86.6.10 GetVideoModeData

Synopsis: Get the specifications for a video mode.

Declaration: `function GetVideoModeData(Index: Word; var Data: TVideoMode) : Boolean`

Visibility: default

Description: `GetVideoModeData` returns the characteristics of the `Index`-th video mode in `Data`. `Index` is zero based, and has a maximum value of `GetVideoModeCount-1`. If the current driver does not support setting of modes (`GetVideoModeCount=1`) and `Index` is zero, the current mode is returned.

The function returns `True` if the mode data was retrieved successfully, `False` otherwise.

For an example, see GetVideoModeCount (2246).

Errors: In case `Index` has a wrong value, `False` is returned.

See also: GetVideoModeCount (2246), SetVideoMode (2250), GetVideoMode (2245)

86.6.11 InitVideo

Synopsis: Initialize video driver.

Declaration: `procedure InitVideo`

Visibility: default

Description: `InitVideo` Initializes the video subsystem. If the video system was already initialized, it does nothing. After the driver has been initialized, the `VideoBuf` and `OldVideoBuf` pointers are

initialized, based on the `ScreenWidth` and `ScreenHeight` variables. When this is done, the screen is cleared.

For an example, see most other functions.

Errors: if the driver fails to initialize, the `ErrorCode` variable is set.

See also: DoneVideo (2242)

86.6.12 LockScreenUpdate

Synopsis: Prevent further screen updates.

Declaration: `procedure LockScreenUpdate`

Visibility: default

Description: `LockScreenUpdate` increments the screen update lock count with one. As long as the screen update lock count is not zero, UpdateScreen (2251) will not actually update the screen.

This function can be used to optimize screen updating: If a lot of writing on the screen needs to be done (by possibly unknown functions), calling `LockScreenUpdate` before the drawing, and UnlockScreenUpdate (2250) after the drawing, followed by a UpdateScreen (2251) call, all writing will be shown on screen at once.

For an example, see GetLockScreenCount (2244).

Errors: None.

See also: UpdateScreen (2251), UnlockScreenUpdate (2250), GetLockScreenCount (2244)

86.6.13 SetCursorPos

Synopsis: Set write cursor position.

Declaration: `procedure SetCursorPos(NewCursorX: Word; NewCursorY: Word)`

Visibility: default

Description: `SetCursorPos` positions the cursor on the given position: Column `NewCursorX` and row `NewCursorY`. The origin of the screen is the upper left corner, and has coordinates `(0,0)`.

The current position is stored in the `CursorX` and `CursorY` variables.

Errors: None.

See also: SetCursorType (2249)

Listing: ./examples/videoex/ex2.pp

```
program example2;

uses video,keyboard;

Var
  P,PP,D : Integer;
  K: TKeyEvent;

  Procedure PutSquare (P : INteger; C : Char);

  begin
```

```
    VideoBuf^[P]:=Ord(C)+($07 shl 8);
    VideoBuf^[P+ScreenWidth]:=Ord(c)+($07 shl 8);
    VideoBuf^[P+1]:=Ord(c)+($07 shl 8);
    VideoBuf^[P+ScreenWidth+1]:=Ord(c)+($07 shl 8);
  end;

begin
  InitVideo;
  InitKeyBoard;
  P:=0;
  PP:=-1;
  Repeat
    If PP<>-1 then
      PutSquare(PP,' ');
    PutSquare(P,'#');
    SetCursorPos(P Mod ScreenWidth,P div ScreenWidth);
    UpdateScreen(False);
    PP:=P;
    Repeat
      D:=0;
      K:=TranslateKeyEvent(GetKeyEvent);
      Case GetKeyEventCode(K) of
        kbdLeft : If (P Mod ScreenWidth)<>0 then
                    D:=-1;
        kbdUp : If P>=ScreenWidth then
                  D:=-ScreenWidth;
        kbdRight : If ((P+2) Mod ScreenWidth)<>0 then
                    D:=1;
        kbdDown : if (P<(VideoBufSize div 2)-(ScreenWidth*2)) then
                    D:=ScreenWidth;
      end;
    Until (D<>0) or (GetKeyEventChar(K)='q');
    P:=P+D;
  until GetKeyEventChar(K)='q';
  DoneKeyBoard;
  DoneVideo;
end.
```

86.6.14 SetCursorType

Synopsis: Set cursor type.

Declaration: `procedure SetCursorType(NewType: Word)`

Visibility: default

Description: `SetCursorType` sets the cursor to the type specified in `NewType`.

crHiddenHide cursor.

crUnderLineUnderline cursor.

crBlockBlock cursor.

crHalfBlockHalf block cursor.

Errors: None.

See also: SetCursorPos (2248)

86.6.15 SetVideoDriver

Synopsis: Install a new video driver.

Declaration: `function SetVideoDriver(const Driver: TVideoDriver) : Boolean`

Visibility: default

Description: `SetVideoDriver` sets the videodriver to be used to `Driver`. If the current videodriver is initialized (after a call to `InitVideo`) then it does nothing and returns `False`.

A new driver can only be installed if the previous driver was not yet activated (i.e. before a call to InitVideo (2247)) or after it was deactivated (i.e after a call to `DoneVideo`).

For more information about installing a videodriver, see viddriver (2232).

For an example, see the section on writing a custom video driver.

Errors: If the current driver is initialized, then `False` is returned.

See also: viddriver (2232)

86.6.16 SetVideoMode

Synopsis: Set current video mode.

Declaration: `function SetVideoMode(const Mode: TVideoMode) : Boolean`

Visibility: default

Description: `SetVideoMode` sets the video mode to the mode specified in `Mode`:

If the call was successful, then the screen will have `Col` columns and `Row` rows, and will be displaying in color if `Color` is `True`.

The function returns `True` if the mode was set successfully, `False` otherwise.

Note that the video mode may not always be set. E.g. a console on Linux or a telnet session cannot always set the mode. It is important to check the error value returned by this function if it was not successful.

The mode can be set when the video driver has not yet been initialized (i.e. before InitVideo (2247) was called) In that case, the video mode will be stored, and after the driver was initialized, an attempt will be made to set the requested mode. Changing the video driver before the call to `InitVideo` will clear the stored video mode.

To know which modes are valid, use the GetVideoModeCount (2246) and GetVideoModeData (2247) functions. To retrieve the current video mode, use the GetVideoMode (2245) procedure.

Errors: If the specified mode cannot be set, then `errVioNoSuchMode` may be set in `ErrorCode`

See also: GetVideoModeCount (2246), GetVideoModeData (2247), GetVideoMode (2245)

86.6.17 UnlockScreenUpdate

Synopsis: Unlock screen update.

Declaration: `procedure UnlockScreenUpdate`

Visibility: default

Description: `UnlockScreenUpdate` decrements the screen update lock count with one if it is larger than zero. When the lock count reaches zero, the UpdateScreen (2251) will actually update the screen. No screen update will be performed as long as the screen update lock count is nonzero. This mechanism can be used to increase screen performance in case a lot of writing is done.

It is important to make sure that each call to LockScreenUpdate (2248) is matched by exactly one call to `UnlockScreenUpdate`

For an example, see GetLockScreenCount (2244).

Errors: None.

See also: LockScreenUpdate (2248), GetLockScreenCount (2244), UpdateScreen (2251)

86.6.18 UpdateScreen

Synopsis: Update physical screen with internal screen image.

Declaration: `procedure UpdateScreen(Force: Boolean)`

Visibility: default

Description: `UpdateScreen` synchronizes the actual screen with the contents of the `VideoBuf` internal buffer. The parameter `Force` specifies whether the whole screen has to be redrawn (`Force=True`) or only parts that have changed since the last update of the screen.

The `Video` unit keeps an internal copy of the screen as it last wrote it to the screen (in the `OldVideoBuf` array). The current contents of `VideoBuf` are examined to see what locations on the screen need to be updated. On slow terminals (e.g. a Linux telnet session) this mechanism can speed up the screen redraw considerably.

On platforms where mouse cursor visibility is not guaranteed to be preserved during screen updates this routine has to restore the mouse cursor after the update (usually by calling HideMouse from unit Mouse before the real update and ShowMouse afterwards).

For an example, see most other functions.

Errors: None.

See also: ClearScreen (2241)

86.7 TVideoDriver

```
TVideoDriver = record
  InitDriver : procedure;
  DoneDriver : procedure
  ;
  UpdateScreen : procedure(Force: Boolean);
  ClearScreen : procedure
  ;
  SetVideoMode : function(const Mode: TVideoMode) : Boolean;
  GetVideoModeCount
  : function : Word;
  GetVideoModeData : function(Index: Word; var
  Data: TVideoMode) : Boolean;
  SetCursorPos : procedure(NewCursorX
  : Word; NewCursorY: Word);
  GetCursorType : function : Word;
```

```
  SetCursorType
  : procedure(NewType: Word);
  GetCapabilities : function : Word;
  end
```

`TVideoDriver` record can be used to install a custom video driver, with the SetVideoDriver (2250) call.

An explanation of all fields can be found there.

86.8 TVideoMode

```
TVideoMode = record
  Col : Word;
  Row : Word;
  Color : Boolean
  ;
end
```

The `TVideoMode` record describes a videomode. Its fields are self-explaining: `Col,Row` describe the number of columns and rows on the screen for this mode. `Color` is `True` if this mode supports colors, or `False` if not.

Chapter 87

Reference for unit 'WinCRT'

87.1 Used units

Table 87.1: Used units by unit 'WinCRT'

Name	Page
System	1362

87.2 Overview

The wincrt unit provides some auxiliary routines for use with the graph (863) unit, namely keyboard support. It has no connection with the crt (571) unit, nor with the Turbo-Pascal for Windows WinCrt unit. As such, it should not be used by end users. Refer to the crt (571) unit instead.

87.3 Constants, types and variables

87.3.1 Variables

```
directvideo : Boolean
```

On windows, this variable is ignored.

```
lastmode : Word
```

Is supposed to contain the last used video mode, but is actually unused.

87.4 Procedures and functions

87.4.1 delay

Synopsis: Pause program execution.

Declaration: `procedure delay(ms: Word)`

Visibility: default

Description: `Delay` stops program execution for the indicated number `ms` of milliseconds.

See also: sound (2254), nosound (2254)

87.4.2 keypressed

Synopsis: Check if a key was pressed.

Declaration: `function keypressed : Boolean`

Visibility: default

Description: `KeyPressed` returns `True` if the user pressed a key, or `False` if not. It does not wait for the user to press a key.

See also: readkey (2254)

87.4.3 nosound

Synopsis: Stop the speaker.

Declaration: `procedure nosound`

Visibility: default

Description: `NoSound` does nothing, windows does not support this.

See also: sound (2254)

87.4.4 readkey

Synopsis: Read a key from the keyboard.

Declaration: `function readkey : Char`

Visibility: default

Description: `ReadKey` reads a key from the keyboard, and returns the ASCII value of the key, or the scancode of the key in case it is a special key.

The function waits until a key is pressed.

See also: KeyPressed (2254)

87.4.5 sound

Synopsis: Sound PC speaker.

Declaration: `procedure sound(hz: Word)`

Visibility: default

Description: `Sound` sounds the PC speaker. It emits a tone with frequency `Hz` for 500 milliseconds. (the time argument is required by the windows API)

See also: nosound (2254)

87.4.6 textmode

Synopsis: Set indicated text mode.

Declaration: `procedure textmode(mode: Integer)`

Visibility: default

Description: `TextMode` does nothing.

Chapter 88

Reference for unit 'WinDirs'

88.1 Used units

Table 88.1: Used units by unit 'WinDirs'

Name	Page
System	1362

88.2 Overview

This unit contains only one function: GetWindowsSpecialDir (2270). It is a simple pascal wrapper around the shfolder.dll library, and is available under Windows only.

88.3 Constants, types and variables

88.3.1 Constants

`CSIDL_ADMINTOOLS = $0030`

ID for the personal "Start Menu\Programs\Administrative tools" folder (In the user's personal folder).

`CSIDL_APPDATA = $001A`

ID for the roaming "Application Data" folder (in the user's personal folder).

`CSIDL_CDBURN_AREA = $003B`

ID for the personal "Local Settings\Application Data\Microsoft\CD Burning" folder (Under "All users" folder).

`CSIDL_COMMON_ADMINTOOLS = $002F`

ID for the common "Start Menu\Programs\Administrative tools" folder (Under "All users" folder).

CSIDL_COMMON_APPDATA = $0023

ID for the common "Application Data" folder (Under "All users" folder).

CSIDL_COMMON_DESKTOPDIRECTORY = $0019

ID for the public "Desktop" folder (Under "All users" folder).

CSIDL_COMMON_DOCUMENTS = $002E

ID for the common "Documents" folder (Under "All users" folder).

CSIDL_COMMON_FAVORITES = $001F

ID for the public "Favorites" folder (Under "All users" folder).

CSIDL_COMMON_MUSIC = $0035

ID for the common "Documents\My Music" folder (Under "All users" folder).

CSIDL_COMMON_PICTURES = $0036

ID for the common "Documents\My Pictures" folder (Under "All users" folder).

CSIDL_COMMON_PROGRAMS = $0017

ID for the public "Programs" folder (Under "All users" Start menu folder).

CSIDL_COMMON_STARTMENU = $0016

ID for the public "Start Menu" folder (Under "All users").

CSIDL_COMMON_STARTUP = $0018

ID for the public "Startup" folder (Under "All users\Start menu\Programs" folder).

CSIDL_COMMON_TEMPLATES = $002D

ID for the common "Templates" folder (Under "All users" folder).

CSIDL_COMMON_VIDEO = $0037

ID for the common "Documents\My Video" folder (Under "All users" folder).

CSIDL_COOKIES = $0021

ID for the "Cookies" folder (in the user's personal folder).

CSIDL_DESKTOPDIRECTORY = $0010

ID for the "Desktop" folder (in the user's personal folder).

```
CSIDL_FAVORITES = $0006
```

ID for the "Favourites" folder (in the user's personal folder).

```
CSIDL_FLAG_CREATE = $8000
```

Flag to specify if the requested folder must be created if it didn't exist yet.

```
CSIDL_FONTS = $0014
```

```
CSIDL_HISTORY = $0022
```

ID for the "History" folder (in the user's personal folder).

```
CSIDL_INTERNET_CACHE = $0020
```

ID for the "Temporary Internet Files" folder (in the user's personal folder).

```
CSIDL_LOCAL_APPDATA = $001C
```

ID for the local "Application Data" folder (in the user's personal folder).

```
CSIDL_MYMUSIC = $000D
```

ID for the "My Music" folder (in the user's personal folder).

```
CSIDL_MYPICTURES = $0027
```

ID for the "My Pictures" folder (in the user's personal folder).

```
CSIDL_MYVIDEO = $000E
```

ID for the "My Videos" folder (in the user's personal folder).

```
CSIDL_NETHOOD = $0013
```

ID for the "Nethood" folder (in the user's personal folder).

```
CSIDL_PERSONAL = $0005
```

ID for the "My Documents" folder (in the user's personal folder).

```
CSIDL_PRINTHOOD = $001B
```

ID for the "Printhood" folder (in the user's personal folder).

```
CSIDL_PROFILE = $0028
```

ID for the user's personal folder.

```
CSIDL_PROFILES = $003E
```

ID for the parent folder for the user's folders.

```
CSIDL_PROGRAMS = $0002
```

ID for the "Program files" folder.

```
CSIDL_PROGRAM_FILES = $0026
```

ID for the "Program Files" folder (alias for `CSIDL_PROGRAMS`).

```
CSIDL_PROGRAM_FILES_COMMON = $002B
```

ID for the Common folder under the "Program Files" folder.

```
CSIDL_RECENT = $0008
```

ID for the "Recent" folder (in the user's personal folder).

```
CSIDL_SENDTO = $0009
```

ID for the "Send to" folder (in the user's personal folder).

```
CSIDL_STARTMENU = $000B
```

ID for the "Start menu" folder (in the user's personal folder).

```
CSIDL_STARTUP = $0007
```

ID for the "Startup" menu folder (in the user's personal folder).

```
CSIDL_SYSTEM = $0025
```

ID for the Windows System32 dir.

```
CSIDL_TEMPLATES = $0015
```

ID for the "Templates" folder (in the user's personal folder).

```
CSIDL_WINDOWS = $0024
```

ID for the Windows installation dir.

```
FOLDERID_AccountPictures : TGuid =
  '{008CA0B1-55B4-4C56-B8A8-4DE4B299D3BE}'


FOLDERID_AddNewPrograms : TGuid =
  '{DE61D971-5EBC-4F02-A3A9-6C82895E5C04}'
```

```
FOLDERID_AdminTools : TGuid =
  '{724EF170-A42D-4FEF-9F26-B60E846FBA4F}'


FOLDERID_AllAppMods : TGuid =
  '{7AD67899-66AF-43BA-9156-6AAD42E6C596}'


FOLDERID_AppCaptures : TGuid =
  '{EDC0FE71-98D8-4F4A-B920-C8DC133CB165}'


FOLDERID_AppDataDesktop : TGuid =
  '{B2C5E279-7ADD-439F-B28C-C41FE1BBF672}'


FOLDERID_AppDataDocuments : TGuid =
  '{7BE16610-1F7F-44AC-BFF0-83E15F2FFCA1}'


FOLDERID_AppDataFavorites : TGuid =
  '{7CFBEFBC-DE1F-45AA-B843-A542AC536CC9}'


FOLDERID_AppDataProgramData : TGuid =
  '{559D40A3-A036-40FA-AF61-84CB430A4D34}'


FOLDERID_ApplicationShortcuts : TGuid =
  '{A3918781-E5F2-4890-B3D9-A7E54332328C}'


FOLDERID_AppsFolder : TGuid =
  '{1E87508D-89C2-42F0-8A7E-645A0F50CA58}'


FOLDERID_AppUpdates : TGuid =
  '{A305CE99-F527-492B-8B1A-7E76FA98D6E4}'


FOLDERID_CameraRoll : TGuid =
  '{AB5FB87B-7CE2-4F83-915D-550846C9537B}'


FOLDERID_CameraRollLibrary : TGuid =
  '{2B20DF75-1EDA-4039-8097-38798227D5B7}'


FOLDERID_CDBurning : TGuid =
  '{9E52AB10-F80D-49DF-ACB8-4330F5687855}'


FOLDERID_ChangeRemovePrograms : TGuid =
  '{DF7266AC-9274-4867-8D55-3BD661DE872D}'
```

```
FOLDERID_CommonAdminTools : TGuid =
  '{D0384E7D-BAC3-4797-8F14-CBA229B392B5}'


FOLDERID_CommonOEMLinks : TGuid =
  '{C1BAE2D0-10DF-4334-BEDD-7AA20B227A9D}'


FOLDERID_CommonPrograms : TGuid =
  '{0139D44E-6AFE-49F2-8690-3DAFCAE6FFB8}'


FOLDERID_CommonStartMenu : TGuid =
  '{A4115719-D62E-491D-AA7C-E74B8BE3B067}'


FOLDERID_CommonStartMenuPlaces : TGuid =
  '{A440879F-87A0-4F7D-B700-0207B966194A}'


FOLDERID_CommonStartup : TGuid =
  '{82A5EA35-D9CD-47C5-9629-E15D2F714E6E}'


FOLDERID_CommonTemplates : TGuid =
  '{B94237E7-57AC-4347-9151-B08C6C32D1F7}'


FOLDERID_ComputerFolder : TGuid =
  '{0AC0837C-BBF8-452A-850D-79D08E667CA7}'


FOLDERID_ConflictFolder : TGuid =
  '{4BFEFB45-347D-4006-A5BE-AC0CB0567192}'


FOLDERID_ConnectionsFolder : TGuid =
  '{6F0CD92B-2E97-45D1-88FF-B0D186B8DEDD}'


FOLDERID_Contacts : TGuid =
  '{56784854-C6CB-462B-8169-88E350ACB882}'


FOLDERID_ControlPanelFolder : TGuid =
  '{82A74AEB-AEB4-465C-A014-D097EE346D63}'


FOLDERID_Cookies : TGuid =
  '{2B0F765D-C0E9-4171-908E-08A611B84FF6}'


FOLDERID_CurrentAppMods : TGuid =
  '{3DB40B20-2A30-4DBE-917E-771DD21DD099}'
```

```
FOLDERID_Desktop : TGuid =
  '{B4BFCC3A-DB2C-424C-B029-7FE99A87C641}'


FOLDERID_DevelopmentFiles : TGuid =
  '{DBE8E08E-3053-4BBC-B183-2A7B2B191E59}'


FOLDERID_Device : TGuid =
  '{1C2AC1DC-4358-4B6C-9733-AF21156576F0}'


FOLDERID_DeviceMetadataStore : TGuid =
  '{5CE4A5E9-E4EB-479D-B89F-130C02886155}'


FOLDERID_Documents : TGuid =
  '{FDD39AD0-238F-46AF-ADB4-6C85480369C7}'


FOLDERID_DocumentsLibrary : TGuid =
  '{7B0DB17D-9CD2-4A93-9733-46CC89022E7C}'


FOLDERID_Downloads : TGuid =
  '{374DE290-123F-4565-9164-39C4925E467B}'


FOLDERID_Favorites : TGuid =
  '{1777F761-68AD-4D8A-87BD-30B759FA33DD}'


FOLDERID_Fonts : TGuid = '{FD228CB7-AE11-4AE3-864C-16F3910AB8FE}'


FOLDERID_Games : TGuid = '{CAC52C1A-B53D-4EDC-92D7-6B2E8AC19434}'


FOLDERID_GameTasks : TGuid =
  '{054FAE61-4DD8-4787-80B6-090220C4B700}'


FOLDERID_History : TGuid =
  '{D9DC8A3B-B784-432E-A781-5A1130A75963}'


FOLDERID_HomeGroup : TGuid =
  '{52528A6B-B9E3-4ADD-B60D-588C2DBA842D}'


FOLDERID_HomeGroupCurrentUser : TGuid =
  '{9B74B6A3-0DFD-4F11-9E78-5F7800F2E772}'
```

```
FOLDERID_ImplicitAppShortcuts : TGuid =
  '{BCB5256F-79F6-4CEE-B725-DC34E402FD46}'


FOLDERID_InternetCache : TGuid =
  '{352481E8-33BE-4251-BA85-6007CAEDCF9D}'


FOLDERID_InternetFolder : TGuid =
  '{4D9F7874-4E0C-4904-967B-40B0D20C3E4B}'


FOLDERID_Libraries : TGuid =
  '{1B3EA5DC-B587-4786-B4EF-BD1DC332AEAE}'


FOLDERID_Links : TGuid = '{BFB9D5E0-C6A9-404C-B2B2-AE6DB6AF4968}'


FOLDERID_LocalAppData : TGuid =
  '{F1B32785-6FBA-4FCF-9D55-7B8E7F157091}'


FOLDERID_LocalAppDataLow : TGuid =
  '{A520A1A4-1780-4FF6-BD18-167343C5AF16}'


FOLDERID_LocalDocuments : TGuid =
  '{F42EE2D3-909F-4907-8871-4C22FC0BF756}'


FOLDERID_LocalDownloads : TGuid =
  '{7D83EE9B-2244-4E70-B1F5-5393042AF1E4}'


FOLDERID_LocalizedResourcesDir : TGuid =
  '{2A00375E-224C-49DE-B8D1-440DF7EF3DDC}'


FOLDERID_LocalMusic : TGuid =
  '{A0C69A99-21C8-4671-8703-7934162FCF1D}'


FOLDERID_LocalPictures : TGuid =
  '{0DDD015D-B06C-45D5-8C4C-F59713854639}'


FOLDERID_LocalVideos : TGuid =
  '{35286A68-3C57-41A1-BBB1-0EAE73D76C95}'


FOLDERID_Music : TGuid = '{4BD8D571-6D19-48D3-BE97-422220080E43}'
```

```
FOLDERID_MusicLibrary : TGuid =
  '{2112AB0A-C86A-4FFE-A368-0DE96E47012E}'


FOLDERID_NetHood : TGuid =
  '{C5ABBF53-E17F-4121-8900-86626FC2C973}'


FOLDERID_NetworkFolder : TGuid =
  '{D20BEEC4-5CA8-4905-AE3B-BF251EA09B53}'


FOLDERID_Objects3D : TGuid =
  '{31C0DD25-9439-4F12-BF41-7FF4EDA38722}'


FOLDERID_OneDrive : TGuid =
  '{A52BBA46-E9E1-435F-B3D9-28DAA648C0F6}'


FOLDERID_OriginalImages : TGuid =
  '{2C36C0AA-5812-4B87-BFD0-4CD0DFB19B39}'


FOLDERID_PhotoAlbums : TGuid =
  '{69D2CF90-FC33-4FB7-9A0C-EBB0F0FCB43C}'


FOLDERID_Pictures : TGuid =
  '{33E28130-4E1E-4676-835A-98395C3BC3BB}'


FOLDERID_PicturesLibrary : TGuid =
  '{A990AE9F-A03B-4E80-94BC-9912D7504104}'


FOLDERID_Playlists : TGuid =
  '{DE92C1C7-837F-4F69-A3BB-86E631204A23}'


FOLDERID_PrintersFolder : TGuid =
  '{76FC4E2D-D6AD-4519-A663-37BD56068185}'


FOLDERID_PrintHood : TGuid =
  '{9274BD8D-CFD1-41C3-B35E-B13F55A758F4}'


FOLDERID_Profile : TGuid =
  '{5E6C858F-0E22-4760-9AFE-EA3317B67173}'


FOLDERID_ProgramData : TGuid =
  '{62AB5D82-FDC1-4DC3-A9DD-070D1D495D97}'
```

```
FOLDERID_ProgramFiles : TGuid =
  '{905E63B6-C1BF-494E-B29C-65B732D3D21A}'


FOLDERID_ProgramFilesCommon : TGuid =
  '{F7F1ED05-9F6D-47A2-AAAE-29D317C6F066}'


FOLDERID_ProgramFilesCommonX64 : TGuid =
  '{6365D5A7-0F0D-45E5-87F6-0DA56B6A4F7D}'


FOLDERID_ProgramFilesCommonX86 : TGuid =
  '{DE974D24-D9C6-4D3E-BF91-F4455120B917}'


FOLDERID_ProgramFilesX64 : TGuid =
  '{6D809377-6AF0-444B-8957-A3773F02200E}'


FOLDERID_ProgramFilesX86 : TGuid =
  '{7C5A40EF-A0FB-4BFC-874A-C0F2E0B9FA8E}'


FOLDERID_Programs : TGuid =
  '{A77F5D77-2E2B-44C3-A6A2-ABA601054A51}'


FOLDERID_Public : TGuid =
  '{DFDF76A2-C82A-4D63-906A-5644AC457385}'


FOLDERID_PublicDesktop : TGuid =
  '{C4AA340D-F20F-4863-AFEF-F87EF2E6BA25}'


FOLDERID_PublicDocuments : TGuid =
  '{ED4824AF-DCE4-45A8-81E2-FC7965083634}'


FOLDERID_PublicDownloads : TGuid =
  '{3D644C9B-1FB8-4F30-9B45-F670235F79C0}'


FOLDERID_PublicGameTasks : TGuid =
  '{DEBF2536-E1A8-4C59-B6A2-414586476AEA}'


FOLDERID_PublicLibraries : TGuid =
  '{48DAF80B-E6CF-4F4E-B800-0E69D84EE384}'


FOLDERID_PublicMusic : TGuid =
  '{3214FAB5-9757-4298-BB61-92A9DEAA44FF}'
```

```
FOLDERID_PublicPictures : TGuid =
  '{B6EBFB86-6907-413C-9AF7-4FC2ABF07CC5}'


FOLDERID_PublicRingtones : TGuid =
  '{E555AB60-153B-4D17-9F04-A5FE99FC15EC}'


FOLDERID_PublicUserTiles : TGuid =
  '{0482AF6C-08F1-4C34-8C90-E17EC98B1E17}'


FOLDERID_PublicVideos : TGuid =
  '{2400183A-6185-49FB-A2D8-4A392A602BA3}'


FOLDERID_QuickLaunch : TGuid =
  '{52A4F021-7B75-48A9-9F6B-4B87A210BC8F}'


FOLDERID_Recent : TGuid =
  '{AE50C081-EBD2-438A-8655-8A092E34987A}'


FOLDERID_RecordedCalls : TGuid =
  '{2F8B40C2-83ED-48EE-B383-A1F157EC6F9A}'


FOLDERID_RecordedTVLibrary : TGuid =
  '{1A6FDBA2-F42D-4358-A798-B74D745926C5}'


FOLDERID_RecycleBinFolder : TGuid =
  '{B7534046-3ECB-4C18-BE4E-64CD4CB7D6AC}'


FOLDERID_ResourceDir : TGuid =
  '{8AD10C31-2ADB-4296-A8F7-E4701232C972}'


FOLDERID_RetailDemo : TGuid =
  '{12D4C69E-24AD-4923-BE19-31321C43A767}'


FOLDERID_Ringtones : TGuid =
  '{C870044B-F49E-4126-A9C3-B52A1FF411E8}'


FOLDERID_RoamedTileImages : TGuid =
  '{AAA8D5A5-F1D6-4259-BAA8-78E7EF60835E}'


FOLDERID_RoamingAppData : TGuid =
  '{3EB685DB-65F9-4CF6-A03A-E3EF65729F3D}'
```

```
FOLDERID_RoamingTiles : TGuid =
  '{00BCFC5A-ED94-4E48-96A1-3F6217F21990}'


FOLDERID_SampleMusic : TGuid =
  '{B250C668-F57D-4EE1-A63C-290EE7D1AA1F}'


FOLDERID_SamplePictures : TGuid =
  '{C4900540-2379-4C75-844B-64E6FAF8716B}'


FOLDERID_SamplePlaylists : TGuid =
  '{15CA69B3-30EE-49C1-ACE1-6B5EC372AFB5}'


FOLDERID_SampleVideos : TGuid =
  '{859EAD94-2E85-48AD-A71A-0969CB56A6CD}'


FOLDERID_SavedGames : TGuid =
  '{4C5C32FF-BB9D-43B0-B5B4-2D72E54EAAA4}'


FOLDERID_SavedPictures : TGuid =
  '{3B193882-D3AD-4EAB-965A-69829D1FB59F}'


FOLDERID_SavedPicturesLibrary : TGuid =
  '{E25B5812-BE88-4BD9-94B0-29233477B6C3}'


FOLDERID_SavedSearches : TGuid =
  '{7D1D3A04-DEBB-4115-95CF-2F29DA2920DA}'


FOLDERID_Screenshots : TGuid =
  '{B7BEDE81-DF94-4682-A7D8-57A52620B86F}'


FOLDERID_SearchHistory : TGuid =
  '{0D4C3DB6-03A3-462F-A0E6-08924C41B5D4}'


FOLDERID_SearchHome : TGuid =
  '{190337D1-B8CA-4121-A639-6D472D16972A}'


FOLDERID_SearchTemplates : TGuid =
  '{7E636BFE-DFA9-4D5E-B456-D7B39851D8A9}'


FOLDERID_SEARCH_CSC : TGuid =
  '{EE32E446-31CA-4ABA-814F-A5EBD2FD6D5E}'
```

```
FOLDERID_SEARCH_MAPI : TGuid =
  '{98EC0E18-2098-4D44-8644-66979315A281}'


FOLDERID_SendTo : TGuid =
  '{8983036C-27C0-404B-8F08-102D10DCFD74}'


FOLDERID_SidebarDefaultParts : TGuid =
  '{7B396E54-9EC5-4300-BE0A-2482EBAE1A26}'


FOLDERID_SidebarParts : TGuid =
  '{A75D362E-50FC-4FB7-AC2C-A8BEAA314493}'


FOLDERID_SkyDrive : TGuid =
  '{A52BBA46-E9E1-435F-B3D9-28DAA648C0F6}'


FOLDERID_SkyDriveCameraRoll : TGuid =
  '{767E6811-49CB-4273-87C2-20F355E1085B}'


FOLDERID_SkyDriveDocuments : TGuid =
  '{24D89E24-2F19-4534-9DDE-6A6671FBB8FE}'


FOLDERID_SkyDriveMusic : TGuid =
  '{C3F2459E-80D6-45DC-BFEF-1F769F2BE730}'


FOLDERID_SkyDrivePictures : TGuid =
  '{339719B5-8C47-4894-94C2-D8F77ADD44A6}'


FOLDERID_StartMenu : TGuid =
  '{625B53C3-AB48-4EC1-BA1F-A1EF4146FC19}'


FOLDERID_StartMenuAllPrograms : TGuid =
  '{F26305EF-6948-40B9-B255-81453D09C785}'


FOLDERID_Startup : TGuid =
  '{B97D20BB-F46A-4C97-BA10-5E3608430854}'


FOLDERID_SyncManagerFolder : TGuid =
  '{43668BF8-C14E-49B2-97C9-747784D784B7}'


FOLDERID_SyncResultsFolder : TGuid =
  '{289A9A43-BE44-4057-A41B-587A76D7E7F9}'
```

```
FOLDERID_SyncSetupFolder : TGuid =
  '{0F214138-B1D3-4A90-BBA9-27CBC0C5389A}'


FOLDERID_System : TGuid =
  '{1AC14E77-02E7-4E5D-B744-2EB1AE5198B7}'


FOLDERID_SystemX86 : TGuid =
  '{D65231B0-B2F1-4857-A4CE-A8E7C6EA7D27}'


FOLDERID_Templates : TGuid =
  '{A63293E8-664E-48DB-A079-DF759E0509F7}'


FOLDERID_UserPinned : TGuid =
  '{9E3995AB-1F9C-4F13-B827-48B24B6C7174}'


FOLDERID_UserProfiles : TGuid =
  '{0762D272-C50A-4BB0-A382-697DCD729B80}'


FOLDERID_UserProgramFiles : TGuid =
  '{5CD7AEE2-2219-4A67-B85D-6C9CE15660CB}'


FOLDERID_UserProgramFilesCommon : TGuid =
  '{BCBD3057-CA5C-4622-B42D-BC56DB0AE516}'


FOLDERID_UsersFiles : TGuid =
  '{F3CE0F7C-4901-4ACC-8648-D5D44B04EF8F}'


FOLDERID_UsersLibraries : TGuid =
  '{A302545D-DEFF-464B-ABE8-61C8648D939B}'


FOLDERID_Videos : TGuid =
  '{18989B1D-99B5-455B-841C-AB7C74E4DDFC}'


FOLDERID_VideosLibrary : TGuid =
  '{491E922F-5643-4AF4-A7EB-4E7A138D8174}'


FOLDERID_Windows : TGuid =
  '{F38BF404-1D43-42F2-9305-67DE0B28FC23}'


KF_FLAG_CREATE = DWORD($8000)


KF_FLAG_DEFAULT = DWORD(0)
```

88.4 Procedures and functions

88.4.1 ConvertCSIDLtoFOLDERID

Declaration:
```
function ConvertCSIDLtoFOLDERID(CSIDL: Integer; out FOLDERID: TGuid)
                                    : Boolean
```

Visibility: default

88.4.2 ConvertFOLDERIDtoCSIDL

Declaration:
```
function ConvertFOLDERIDtoCSIDL(const FOLDERID: TGuid;
                                  out CSIDL: Integer) : Boolean
```

Visibility: default

88.4.3 GetWindowsSpecialDir

Synopsis: Get the location of a special directory.

Declaration:
```
function GetWindowsSpecialDir(ID: Integer; CreateIfNotExists: Boolean)
                                 : string
function GetWindowsSpecialDir(const GUID: TGuid;
                               CreateIfNotExists: Boolean) : string
```

Visibility: default

Description: `GetWindowsSpecialDir` can be used to to get the path of special folders on the Windows system. The locations are identified using one of the `CSIDL_*` constants. If the ID of a location is or-ed with the `CSIDL_FLAG_CREATE` constant, then the directory will be created if it didn't exist yet (and the user has sufficient rights to do so).

Errors: On error, an empty string is returned.

88.4.4 GetWindowsSpecialDirUnicode

Declaration:
```
function GetWindowsSpecialDirUnicode(ID: Integer;
                                     CreateIfNotExists: Boolean)
                                      : UnicodeString
function GetWindowsSpecialDirUnicode(const GUID: TGuid;
                                     CreateIfNotExists: Boolean)
                                      : UnicodeString
```

Visibility: default

88.4.5 GetWindowsSystemDirectory

Declaration:
```
function GetWindowsSystemDirectory : string
```

Visibility: default

88.4.6 GetWindowsSystemDirectoryUnicode

Declaration: `function GetWindowsSystemDirectoryUnicode : UnicodeString`

Visibility: default

Chapter 89

Reference for unit 'x86'

89.1 Used units

Table 89.1: Used units by unit 'x86'

Name	Page
BaseUnix	146
System	1362

89.2 Overview

The x86 unit contains some of the routines that were present in the 1.0.X Linux unit, and which were Intel (PC) architecture specific.

These calls have been preserved for compatibility, but should be considered deprecated: they are not portable and may not even work on future Linux versions.

89.3 Procedures and functions

89.3.1 fpIOperm

Synopsis: Set permission on IO ports.

Declaration: `function fpIOperm(From: Cardinal; Num: Cardinal; Value: cint) : cint`

Visibility: default

Description: `FpIOperm` sets permissions on `Num` ports starting with port `From` to `Value`. The function returns zero if the call was successful, a nonzero value otherwise.

Note:

- This works ONLY as root.
- Only the first `0x03ff` ports can be set.
- When doing a FpFork (195), the permissions are reset. When doing a FpExecVE (192) they are kept.

Errors: Extended error information can be retrieved with FpGetErrno (198)

89.3.2 fpIoPL

Synopsis: Set I/O privilege level.

Declaration:
```
function fpIoPL(Level: cint) : cint
```

Visibility: default

Description: `FpIoPL` sets the I/O privilege level. It is intended for completeness only, one should normally not use it.

89.3.3 ReadPort

Synopsis: Read data from a PC port.

Declaration:
```
procedure ReadPort(Port: LongInt; var Value: Byte)
procedure ReadPort(Port: LongInt; var Value: LongInt)
procedure ReadPort(Port: LongInt; var Value: Word)
```

Visibility: default

Description: `ReadPort` reads one Byte, Word or Longint from port `Port` into `Value`.

Note that you need permission to read a port. This permission can be set by the root user with the FpIOPerm (2272) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: FpIOPerm (2272), ReadPortB (2273), ReadPortW (2274), ReadPortL (2274), WritePort (2274), WritePortB (2275), WritePortL (2275), WritePortW (2275)

89.3.4 ReadPortB

Synopsis: Read bytes from a PC port.

Declaration:
```
function ReadPortB(Port: LongInt) : Byte
procedure ReadPortB(Port: LongInt; var Buf; Count: LongInt)
```

Visibility: default

Description: The procedural form of `ReadPortB` reads `Count` bytes from port `Port` and stores them in `Buf`. There must be enough memory allocated at `Buf` to store `Count` bytes.

The functional form of `ReadPortB` reads 1 byte from port `B` and returns the byte that was read.

Note that you need permission to read a port. This permission can be set by the root user with the FpIOPerm (2272) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: FpIOPerm (2272), ReadPort (2273), ReadPortW (2274), ReadPortL (2274), WritePort (2274), WritePortB (2275), WritePortL (2275), WritePortW (2275)

89.3.5 ReadPortL

Synopsis: Read longints from a PC port.

Declaration:
```
function ReadPortL(Port: LongInt) : LongInt
procedure ReadPortL(Port: LongInt; var Buf; Count: LongInt)
```

Visibility: default

Description: The procedural form of `ReadPortL` reads `Count` longints from port `Port` and stores them in `Buf`. There must be enough memory allocated at `Buf` to store `Count` Longints.

The functional form of `ReadPortL` reads 1 longint from port `B` and returns the longint that was read.

Note that you need permission to read a port. This permission can be set by the root user with the FpIOPerm (2272) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: FpIOPerm (2272), ReadPort (2273), ReadPortW (2274), ReadPortB (2273), WritePort (2274), WritePortB (2275), WritePortL (2275), WritePortW (2275)

89.3.6 ReadPortW

Synopsis: Read Words from a PC port.

Declaration:
```
function ReadPortW(Port: LongInt) : Word
procedure ReadPortW(Port: LongInt; var Buf; Count: LongInt)
```

Visibility: default

Description: The procedural form of `ReadPortW` reads `Count` words from port `Port` and stores them in `Buf`. There must be enough memory allocated at `Buf` to store `Count` words.

The functional form of `ReadPortW` reads 1 word from port `B` and returns the word that was read.

Note that you need permission to read a port. This permission can be set by the root user with the FpIOPerm (2272) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: FpIOPerm (2272), ReadPort (2273), ReadPortB (2273), ReadPortL (2274), WritePort (2274), WritePortB (2275), WritePortL (2275), WritePortW (2275)

89.3.7 WritePort

Synopsis: Write data to PC port.

Declaration:
```
procedure WritePort(Port: LongInt; Value: Byte)
procedure WritePort(Port: LongInt; Value: LongInt)
procedure WritePort(Port: LongInt; Value: Word)
```

Visibility: default

Description: `WritePort` writes `Value` – 1 byte, Word or longint – to port `Port`.

Remark You need permission to write to a port. This permission can be set with root permission with the FpIOPerm (2272) call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: FpIOPerm (2272), WritePortB (2275), WritePortL (2275), WritePortW (2275), ReadPortB (2273), ReadPortL (2274), ReadPortW (2274)

89.3.8 WritePortB

Synopsis: Write byte to PC port.

Declaration:
```
procedure WritePortB(Port: LongInt; Value: Byte)
procedure WritePortB(Port: LongInt; var Buf; Count: LongInt)
```

Visibility: default

Description: The first form of `WritePortB` writes 1 byte to port `Port`. The second form writes `Count` bytes from `Buf` to port `Port`.

Remark You need permission to write to a port. This permission can be set with root permission with the FpIOPerm (2272) call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: FpIOPerm (2272), WritePort (2274), WritePortL (2275), WritePortW (2275), ReadPortB (2273), ReadPortL (2274), ReadPortW (2274)

89.3.9 WritePortl

Synopsis: Write longint to PC port.

Declaration:
```
procedure WritePortL(Port: LongInt; Value: LongInt)
procedure WritePortl(Port: LongInt; var Buf; Count: LongInt)
```

Visibility: default

Description: The first form of `WritePortB` writes 1 byte to port `Port`. The second form writes `Count` bytes from `Buf` to port `Port`.

Remark You need permission to write to a port. This permission can be set with root permission with the FpIOPerm (2272) call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: FpIOPerm (2272), WritePort (2274), WritePortB (2275), WritePortW (2275), ReadPortB (2273), ReadPortL (2274), ReadPortW (2274)

89.3.10 WritePortW

Synopsis: Write Word to PC port.

Declaration:
```
procedure WritePortW(Port: LongInt; Value: Word)
procedure WritePortW(Port: LongInt; var Buf; Count: LongInt)
```

Visibility: default

Description: The first form of `WritePortB` writes 1 byte to port `Port`. The second form writes `Count` bytes from `Buf` to port `Port`.

Remark You need permission to write to a port. This permission can be set with root permission with the FpIOPerm (2272) call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: FpIOPerm (2272), WritePort (2274), WritePortL (2275), WritePortB (2275), ReadPortB (2273), ReadPortL (2274), ReadPortW (2274)