

RMOL

0.25.3

Generated by Doxygen 1.8.1.2

Wed Aug 29 2012 12:42:08

Contents

1	RMOL Documentation	1
1.1	Getting Started	1
1.2	RMOL at SourceForge	1
1.3	RMOL Development	1
1.4	External Libraries	2
1.5	Support RMOL	2
1.6	About RMOL	2
2	People	2
2.1	Project Admins	2
2.2	Developers	2
2.3	Retired Developers	2
2.4	Contributors	3
2.5	Distribution Maintainers	3
3	Coding Rules	3
3.1	Default Naming Rules for Variables	3
3.2	Default Naming Rules for Functions	3
3.3	Default Naming Rules for Classes and Structures	3
3.4	Default Naming Rules for Files	3
3.5	Default Functionality of Classes	4
4	Copyright and License	4
4.1	GNU LESSER GENERAL PUBLIC LICENSE	4
4.1.1	Version 2.1, February 1999	4
4.2	Preamble	4
4.3	TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION	5
4.3.1	NO WARRANTY	9
4.3.2	END OF TERMS AND CONDITIONS	9
4.4	How to Apply These Terms to Your New Programs	9
5	Documentation Rules	10
5.1	General Rules	10
5.2	File Header	11
5.3	Grouping Various Parts	11
6	Main features	12
6.1	Optimisation features	12
6.2	Unconstraining	12
6.3	Forecasting features	12

6.4	Overbooking features	12
6.5	Other features	12
7	Make a Difference	12
8	Make a new release	13
8.1	Introduction	13
8.2	Initialisation	13
8.3	Release branch maintenance	13
8.4	Commit and publish the release branch	14
8.5	Create source packages (tar-balls)	14
8.6	Upload the HTML documentation to SourceForge	14
8.7	Generate the RPM packages	15
8.8	Update distributed change log	15
8.9	Create the binary package, including the documentation	15
8.10	Upload the files to SourceForge	15
8.11	Make a new post	15
8.12	Send an email on the announcement mailing-list	15
9	Installation	16
9.1	Table of Contents	16
9.2	Fedora/RedHat Linux distributions	16
9.3	RMOL Requirements	16
9.4	Basic Installation	17
9.5	Compilers and Options	18
9.6	Compiling For Multiple Architectures	18
9.7	Installation Names	18
9.8	Optional Features	19
9.9	Particular systems	19
9.10	Specifying the System Type	20
9.11	Sharing Defaults	20
9.12	Defining Variables	21
9.13	'cmake' Invocation	21
10	Linking with RMOL	25
10.1	Table of Contents	25
10.2	Introduction	25
10.3	Using the pkg-config command	25
10.4	Using the rmol-config script	25
10.5	M4 macro for the GNU Autotools	26
10.6	Using RMOL with dynamic linking	26

11 Test Rules	26
11.1 The Test File	26
11.2 The Reference File	26
11.3 Testing IT++ Library	26
12 Users Guide	27
12.1 Table of Contents	27
12.2 Introduction	27
12.3 Get Started	27
12.3.1 Get the RMOL library	27
12.3.2 Build the RMOL project	27
12.3.3 Build and Run the Tests	27
12.3.4 Install the RMOL Project (Binaries, Documentation)	27
12.4 Exploring the Predefined BOM Tree	27
12.4.1 Forecaster BOM Tree	27
12.4.2 Optimiser BOM Tree	28
12.5 Extending the BOM Tree	28
13 Supported Systems	28
13.1 Table of Contents	28
13.2 Introduction	28
13.3 RMOL 0.23.x	29
13.3.1 Linux Systems	29
13.3.2 Windows Systems	32
13.3.3 Unix Systems	35
14 RMOL Supported Systems (Previous Releases)	35
14.1 RMOL 3.9.1	35
14.2 RMOL 3.9.0	35
14.3 RMOL 3.8.1	35
15 Tutorials	35
15.1 Table of Contents	35
15.2 Introduction	36
15.2.1 Preparing the StdAir Project for Development	36
15.3 Build a Predefined BOM Tree	36
15.3.1 Instantiate the BOM Root Object	36
15.3.2 Instantiate the (Airline) Inventory Object	36
15.3.3 Link the Inventory Object with the BOM Root	36
15.3.4 Build Another Airline Inventory	37
15.3.5 Dump The BOM Tree Content	37

15.3.6 Result of the Tutorial Program	37
15.4 Extend the Pre-Defined BOM Tree	38
15.4.1 Extend an Airline Inventory Object	38
15.4.2 Build the Specific BOM Objects	38
15.4.3 Result of the Tutorial Program	39
16 Command-Line Test to Demonstrate How To Test the RMOL Project	39
17 Command-Line Test to Demonstrate How To Test the RMOL Project	43
18 Command-Line Test to Demonstrate How To Test the RMOL Project	43
19 Command-Line Test to Demonstrate How To Test the RMOL Project	46
20 Namespace Index	47
20.1 Namespace List	47
21 Class Index	47
21.1 Class Hierarchy	47
22 Class Index	49
22.1 Class List	49
23 File Index	50
23.1 File List	50
24 Namespace Documentation	52
24.1 RMOL Namespace Reference	52
24.1.1 Typedef Documentation	53
24.1.2 Variable Documentation	54
24.2 stdair Namespace Reference	55
24.2.1 Detailed Description	55
25 Class Documentation	56
25.1 CmdAbstract Class Reference	56
25.2 RMOL::DefaultDCPList Struct Reference	56
25.2.1 Detailed Description	56
25.2.2 Member Function Documentation	56
25.3 RMOL::DefaultMap Struct Reference	56
25.3.1 Detailed Description	56
25.3.2 Member Function Documentation	57
25.4 RMOL::DemandGeneratorList Class Reference	57
25.4.1 Detailed Description	57
25.4.2 Member Typedef Documentation	57

25.4.3	Constructor & Destructor Documentation	57
25.4.4	Member Function Documentation	58
25.5	RMOL::Detruncator Class Reference	58
25.5.1	Detailed Description	58
25.5.2	Member Function Documentation	58
25.6	RMOL::DPOptimiser Class Reference	59
25.6.1	Detailed Description	59
25.6.2	Member Function Documentation	59
25.7	RMOL::EMDetruncator Class Reference	60
25.7.1	Detailed Description	60
25.7.2	Member Function Documentation	60
25.8	RMOL::Emsr Class Reference	60
25.8.1	Detailed Description	61
25.8.2	Member Function Documentation	61
25.9	RMOL::EmsrUtils Class Reference	61
25.9.1	Detailed Description	61
25.9.2	Member Function Documentation	62
25.10	RMOL::FacRmolServiceContext Class Reference	62
25.10.1	Detailed Description	63
25.10.2	Constructor & Destructor Documentation	63
25.10.3	Member Function Documentation	63
25.11	FacServiceAbstract Class Reference	63
25.12	RMOL::Forecaster Class Reference	64
25.12.1	Detailed Description	64
25.12.2	Member Function Documentation	64
25.13	ForecasterTestSuite Class Reference	64
25.13.1	Detailed Description	65
25.13.2	Constructor & Destructor Documentation	65
25.13.3	Member Function Documentation	65
25.13.4	Member Data Documentation	65
25.14	RMOL::ForecastException Class Reference	65
25.14.1	Detailed Description	66
25.14.2	Constructor & Destructor Documentation	66
25.15	RMOL::GuillotineBlockHelper Class Reference	66
25.15.1	Detailed Description	66
25.15.2	Member Function Documentation	66
25.16	RMOL::HistoricalBooking Struct Reference	67
25.16.1	Detailed Description	67
25.16.2	Constructor & Destructor Documentation	67
25.16.3	Member Function Documentation	68

25.17RMOL::HistoricalBookingHolder Struct Reference	69
25.17.1 Detailed Description	70
25.17.2 Constructor & Destructor Documentation	70
25.17.3 Member Function Documentation	70
25.18RMOL::InventoryParser Class Reference	73
25.18.1 Detailed Description	73
25.18.2 Member Function Documentation	73
25.19RMOL::MCOptimiser Class Reference	73
25.19.1 Detailed Description	74
25.19.2 Member Function Documentation	74
25.20RMOL::OptimisationException Class Reference	74
25.20.1 Detailed Description	75
25.20.2 Constructor & Destructor Documentation	75
25.21RMOL::Optimiser Class Reference	75
25.21.1 Detailed Description	75
25.21.2 Member Function Documentation	75
25.22OptimiseTestSuite Class Reference	76
25.22.1 Detailed Description	77
25.22.2 Constructor & Destructor Documentation	77
25.22.3 Member Function Documentation	77
25.22.4 Member Data Documentation	77
25.23RMOL::OverbookingException Class Reference	78
25.23.1 Detailed Description	78
25.23.2 Constructor & Destructor Documentation	78
25.24RMOL::RMOL_Service Class Reference	78
25.24.1 Detailed Description	79
25.24.2 Constructor & Destructor Documentation	79
25.24.3 Member Function Documentation	80
25.25RMOL::RMOL_ServiceContext Class Reference	84
25.25.1 Detailed Description	85
25.25.2 Friends And Related Function Documentation	85
25.26RootException Class Reference	85
25.27ServiceAbstract Class Reference	85
25.28StructAbstract Class Reference	86
25.29TestFixture Class Reference	86
25.30UnconstrainerTestSuite Class Reference	86
25.30.1 Detailed Description	87
25.30.2 Constructor & Destructor Documentation	87
25.30.3 Member Function Documentation	87
25.30.4 Member Data Documentation	87

25.31 RMOL::UnconstrainingException Class Reference	87
25.31.1 Detailed Description	88
25.31.2 Constructor & Destructor Documentation	88
25.32 RMOL::Utilities Class Reference	88
25.32.1 Detailed Description	88
25.32.2 Member Function Documentation	88
26 File Documentation	89
26.1 doc/local/authors.doc File Reference	89
26.2 doc/local/codingrules.doc File Reference	89
26.3 doc/local/copyright.doc File Reference	89
26.4 doc/local/documentation.doc File Reference	89
26.5 doc/local/features.doc File Reference	89
26.6 doc/local/help_wanted.doc File Reference	89
26.7 doc/local/howto_release.doc File Reference	89
26.8 doc/local/index.doc File Reference	89
26.9 doc/local/installation.doc File Reference	89
26.10 doc/local/linking.doc File Reference	89
26.11 doc/local/test.doc File Reference	89
26.12 doc/local/users_guide.doc File Reference	89
26.13 doc/local/verification.doc File Reference	89
26.14 doc/tutorial/tutorial.doc File Reference	89
26.15 rmol/basic/BasConst.cpp File Reference	89
26.16 BasConst.cpp	90
26.17 rmol/basic/BasConst_Curves.hpp File Reference	91
26.18 BasConst_Curves.hpp	91
26.19 rmol/basic/BasConst_General.hpp File Reference	91
26.20 BasConst_General.hpp	92
26.21 rmol/basic/BasConst_RMOL_Service.hpp File Reference	92
26.22 BasConst_RMOL_Service.hpp	92
26.23 rmol/batches/rmol.cpp File Reference	93
26.23.1 Function Documentation	93
26.23.2 Variable Documentation	94
26.24 rmol.cpp	95
26.25 rmol/bom/BucketHolderTypes.hpp File Reference	98
26.26 BucketHolderTypes.hpp	98
26.27 rmol/bom/DistributionParameterList.hpp File Reference	98
26.28 DistributionParameterList.hpp	99
26.29 rmol/bom/DPOptimiser.cpp File Reference	99
26.30 DPOptimiser.cpp	99

26.31 rmol/bom/DPOptimiser.hpp File Reference	102
26.32 DPOptimiser.hpp	102
26.33 rmol/bom/EMDetruncator.cpp File Reference	103
26.34 EMDetruncator.cpp	103
26.35 rmol/bom/EMDetruncator.hpp File Reference	104
26.36 EMDetruncator.hpp	104
26.37 rmol/bom/Emsr.cpp File Reference	105
26.38 Emsr.cpp	105
26.39 rmol/bom/Emsr.hpp File Reference	107
26.40 Emsr.hpp	107
26.41 rmol/bom/EmsrUtils.cpp File Reference	108
26.42 EmsrUtils.cpp	108
26.43 rmol/bom/EmsrUtils.hpp File Reference	109
26.44 EmsrUtils.hpp	109
26.45 rmol/bom/GuillotineBlockHelper.cpp File Reference	110
26.46 GuillotineBlockHelper.cpp	110
26.47 rmol/bom/GuillotineBlockHelper.hpp File Reference	111
26.48 GuillotineBlockHelper.hpp	111
26.49 rmol/bom/HistoricalBooking.cpp File Reference	112
26.50 HistoricalBooking.cpp	112
26.51 rmol/bom/HistoricalBooking.hpp File Reference	113
26.52 HistoricalBooking.hpp	113
26.53 rmol/bom/HistoricalBookingHolder.cpp File Reference	114
26.54 HistoricalBookingHolder.cpp	114
26.55 rmol/bom/HistoricalBookingHolder.hpp File Reference	118
26.56 HistoricalBookingHolder.hpp	118
26.57 rmol/bom/MCOptimiser.cpp File Reference	119
26.58 MCOptimiser.cpp	119
26.59 rmol/bom/MCOptimiser.hpp File Reference	123
26.60 MCOptimiser.hpp	123
26.61 rmol/bom/old/DemandGeneratorList.cpp File Reference	124
26.62 DemandGeneratorList.cpp	124
26.63 rmol/bom/old/DemandGeneratorList.hpp File Reference	125
26.64 DemandGeneratorList.hpp	125
26.65 rmol/bom/Utilities.cpp File Reference	126
26.66 Utilities.cpp	126
26.67 rmol/bom/Utilities.hpp File Reference	128
26.68 Utilities.hpp	128
26.69 rmol/command/Detruncator.cpp File Reference	128
26.70 Detruncator.cpp	129

26.71 rmol/command/Detruncator.hpp File Reference	136
26.72 Detruncator.hpp	137
26.73 rmol/command/Forecaster.cpp File Reference	138
26.74 Forecaster.cpp	138
26.75 rmol/command/Forecaster.hpp File Reference	149
26.76 Forecaster.hpp	149
26.77 rmol/command/InventoryParser.cpp File Reference	151
26.78 InventoryParser.cpp	151
26.79 rmol/command/InventoryParser.hpp File Reference	153
26.80 InventoryParser.hpp	153
26.81 rmol/command/Optimiser.cpp File Reference	154
26.82 Optimiser.cpp	154
26.83 rmol/command/Optimiser.hpp File Reference	157
26.84 Optimiser.hpp	157
26.85 rmol/config/rmol-paths.hpp File Reference	158
26.85.1 Macro Definition Documentation	158
26.86 rmol-paths.hpp	159
26.87 rmol/factory/FacRmolServiceContext.cpp File Reference	160
26.88 FacRmolServiceContext.cpp	160
26.89 rmol/factory/FacRmolServiceContext.hpp File Reference	161
26.90 FacRmolServiceContext.hpp	161
26.91 rmol/RMOL_Service.hpp File Reference	161
26.92 RMOL_Service.hpp	162
26.93 rmol/RMOL_Types.hpp File Reference	164
26.94 RMOL_Types.hpp	165
26.95 rmol/service/RMOL_Service.cpp File Reference	166
26.96 RMOL_Service.cpp	167
26.97 rmol/service/RMOL_ServiceContext.cpp File Reference	190
26.98 RMOL_ServiceContext.cpp	191
26.99 rmol/service/RMOL_ServiceContext.hpp File Reference	192
26.100 RMOL_ServiceContext.hpp	192
26.101 test/rmol/bomsforforecaster.cpp File Reference	193
26.102 bomsforforecaster.cpp	193
26.103 test/rmol/ForecasterTestSuite.cpp File Reference	196
26.104 ForecasterTestSuite.cpp	196
26.105 test/rmol/ForecasterTestSuite.hpp File Reference	197
26.105.1 Function Documentation	197
26.106 ForecasterTestSuite.hpp	197
26.107 test/rmol/OptimiseTestSuite.cpp File Reference	198
26.108 OptimiseTestSuite.cpp	198

test/rmol/OptimiseTestSuite.hpp File Reference	201
26.109. Function Documentation	201
26.110 OptimiseTestSuite.hpp	201
test/rmol/UnconstrainerTestSuite.cpp File Reference	202
26.112 UnconstrainerTestSuite.cpp	202
test/rmol/UnconstrainerTestSuite.hpp File Reference	202
26.113. Function Documentation	203
26.114 UnconstrainerTestSuite.hpp	203

1 RMOL Documentation

1.1 Getting Started

- [Main features](#)
- [Installation](#)
- [Linking with RMOL](#)
- [Users Guide](#)
- [Tutorials](#)
- [Copyright and License](#)
- [Make a Difference](#)
- [Make a new release](#)
- [People](#)

1.2 RMOL at SourceForge

- [Project page](#)
- [Download RMOL](#)
- [Open a ticket for a bug or feature](#)
- [Mailing lists](#)
- [Forums](#)
 - [Discuss about Development issues](#)
 - [Ask for Help](#)
 - [Discuss RMOL](#)

1.3 RMOL Development

- [Git Repository](#) (Subversion is deprecated)
- [Coding Rules](#)
- [Documentation Rules](#)
- [Test Rules](#)

1.4 External Libraries

- [Boost](#) (C++ STL extensions)
- [Python](#)
- [MySQL client](#)
- [SOI](#) (C++ DB API)

1.5 Support RMOL

1.6 About RMOL

[RMOL](#) is a C++ library of revenue management and optimisation classes and functions. [RMOL](#) mainly targets simulation purposes. [N](#)

[RMOL](#) makes an extensive use of existing open-source libraries for increased functionality, speed and accuracy. In particular [GSL](#) (*GNU Scientific Library*) and [Boost](#) (*C++ Standard Extensions*) libraries are used.

The [RMOL](#) library originates from the department of Operational Research and Innovation at [Amadeus](#), Sophia Antipolis, France. [RMOL](#) is released under the terms of the [GNU Lesser General Public License](#) (LGPLv2.1) for you to enjoy.

[RMOL](#) should work on [GNU/Linux](#), [Sun Solaris](#), Microsoft Windows (with [Cygwin](#), [MinGW/MSYS](#), or [Microsoft Visual C++ .NET](#)) and [Mac OS X](#) operating systems.

Note

(N) - The [RMOL](#) library is **NOT** intended, in any way, to be used by airlines for production systems. If you want to report issue, bug or feature request, or if you just want to give feedback, have a look on the right-hand side of this page for the preferred reporting methods. In any case, please do not contact Amadeus directly for any matter related to [RMOL](#).

2 People

2.1 Project Admins

- Denis Arnaud denis_arnaud@users.sourceforge.net ([N](#))
- Anh Quan Nguyen quannaus@users.sourceforge.net ([N](#))

2.2 Developers

- Anh Quan Nguyen quannaus@users.sourceforge.net ([N](#))
- Denis Arnaud denis_arnaud@users.sourceforge.net ([N](#))
- Nicolas Bondoux nbondoux@users.sourceforge.net ([N](#))

2.3 Retired Developers

- Patrick Grandjean pgrandjean@users.sourceforge.net ([N](#))
- Benoit Lardeux benlardeux@users.sourceforge.net ([N](#))
- Karim Duval duvalkarim@users.sourceforge.net ([N](#))
- Ngoc-Thach Hoang hoangngocthach@users.sourceforge.net ([N](#))
- Son Nguyen Kim snguyenkim@users.sourceforge.net ([N](#))

2.4 Contributors

- Emmanuel Bastien ebastien@users.sourceforge.net (N)
- Christophe Lacombe ddtoof@users.sourceforge.net (N)

2.5 Distribution Maintainers

- **Fedora/RedHat**: Denis Arnaud denis_arnaud@users.sourceforge.net (N)
- **Debian**: Emmanuel Bastien ebastien@users.sourceforge.net (N)

Note

(N) - **Amadeus** employees.

3 Coding Rules

In the following sections we describe the naming conventions which are used for files, classes, structures, local variables, and global variables.

3.1 Default Naming Rules for Variables

Variables names follow Java naming conventions. Examples:

- `lNumberOfPassengers`
- `lSeatAvailability`

3.2 Default Naming Rules for Functions

Function names follow Java naming conventions. Example:

- `int myFunctionName (const int& a, int b)`

3.3 Default Naming Rules for Classes and Structures

Each new word in a class or structure name should always start with a capital letter and the words should be separated with an under-score. Abbreviations are written with capital letters. Examples:

- `MyClassName`
- `MyStructName`

3.4 Default Naming Rules for Files

Files are named after the C++ class names.

Source files are named using `.cpp` suffix, whereas header files end with `.hpp` extension. Examples:

- `FlightDate.hpp`
- `SegmentDate.cpp`

3.5 Default Functionality of Classes

All classes that are configured by input parameters should include:

- default empty constructor
- one or more additional constructor(s) that takes input parameters and initializes the class instance
- setup function, preferably named `'setup'` or `'set_parameters'`

Explicit destructor functions are not required, unless they are needed. It shall not be possible to use any of the other member functions unless the class has been properly initiated with the input parameters.

4 Copyright and License

4.1 GNU LESSER GENERAL PUBLIC LICENSE

4.1.1 Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

4.2 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages—typically libraries—of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

4.3 TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of

any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

1. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- a) The modified work must itself be a software library.
- b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
- c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
- d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

1. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

1. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

1. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

1. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components

(compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

1. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

1. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

1. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

1. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

1. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

1. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among

countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

1. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

1. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

4.3.1 NO WARRANTY

1. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
1. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

4.3.2 END OF TERMS AND CONDITIONS

4.4 How to Apply These Terms to Your New Programs

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the library's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>
```

```
This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.
```

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the library 'Frob' (a library for tweaking knobs) written by James Random Hacker.

<signature of Ty Coon>, 1 April 1990
Ty Coon, President of Vice

That's all there is to it!

Source

5 Documentation Rules

5.1 General Rules

All classes in [RMOL](#) should be properly documented with Doxygen comments in include (.hpp) files. Source (.cpp) files should be documented according to a normal standard for well documented C++ code.

An example of how the interface of a class shall be documented in [RMOL](#) is shown here:

```

/*!
 * \brief Brief description of MyClass here
 *
 * Detailed description of MyClass here. With example code if needed.
 */
class MyClass {
public:
    ///! Default constructor
    MyClass(void) { setup_done = false; }

    /*!
     * \brief Constructor that initializes the class with parameters
     *
     * Detailed description of the constructor here if needed
     *
     * \param[in] param1 Description of \a param1 here
     * \param[in] param2 Description of \a param2 here
     */
    MyClass(TYPE1 param1, TYPE2 param2) { setup(param1, param2); }

    /*!
     * \brief Setup function for MyClass
     *
     * Detailed description of the setup function here if needed
     *
     * \param[in] param1 Description of \a param1 here
     * \param[in] param2 Description of \a param2 here
     */
    void setup(TYPE1 param1, TYPE2 param2);

    /*!
     * \brief Brief description of memberFunction1
     *

```

```

    * Detailed description of memberFunction1 here if needed
    *
    * \param[in]    param1 Description of \a param1 here
    * \param[in]    param2 Description of \a param2 here
    * \param[in,out] param3 Description of \a param3 here
    * \return Description of the return value here
    */
    TYPE4 memberFunction1(TYPE1 param1, TYPE2 param2, TYPE3 &param3);

private:

    bool _setUpDone;          /*!< Variable that checks if the class is properly
                               initialized with parameters */
    TYPE1 _privateVariable1; /*!< Short description of _privateVariable1 here
    TYPE2 _privateVariable2; /*!< Short description of _privateVariable2 here
};

```

5.2 File Header

All files should start with the following header, which include Doxygen's `\file`, `\brief` and `\author` tags, `$Date$` and `$Revisions$` CVS tags, and a common copyright note:

```

/*!
 * \file
 * \brief Brief description of the file here
 * \author Names of the authors who contributed to this code
 * \date Date
 *
 * Detailed description of the file here if needed.
 *
 * -----
 *
 * RMOL - C++ Revenue Management Object Library
 *
 * Copyright (C) 2007-2010 (\see authors file for a list of contributors)
 *
 * \see copyright file for license information
 *
 * -----
 */

```

5.3 Grouping Various Parts

All functions must be added to a Doxygen group in order to appear in the documentation. The following code example defines the group `'my_group'`:

```

/*!
 * \defgroup my_group Brief description of the group here
 *
 * Detailed description of the group here
 */

```

The following example shows how to document the function `myFunction` and how to add it to the group `my_group`:

```

/*!
 * \brief Brief description of myFunction here
 * \ingroup my_group
 *
 * Detailed description of myFunction here
 *
 * \param[in] param1 Description of \a param1 here
 * \param[in] param2 Description of \a param2 here
 * \return Description of the return value here
 */
TYPE3 myFunction(TYPE1 param1, TYPE2 &param2);

```

6 Main features

A short list of the main features of **RMOL** is given below sorted in different categories. Many more features and functions exist and for these we refer to the reference documentation.

6.1 Optimisation features

- **Dynamic Programming (DP)**
- **EMSRa** and **EMSRb**
- Network optimisation with **Linear Programming (LP)**

6.2 Unconstraining

- Inventory censorflag and guillotine
- E-M (Expectation Maximisation)

6.3 Forecasting features

- **Exponential Smoothing**
- **Moving Average**

6.4 Overbooking features

- Cancellations and No-Shows
- Cost-based optimisation
- Service-based optimisation

6.5 Other features

- CSV input file parsing

7 Make a Difference

Do not ask what **RMOL can do for you. Ask what you can do for **RMOL**.**

You can help us to develop the **RMOL** library. There are always a lot of things you can do:

- Start using **RMOL**
- Tell your friends about **RMOL** and help them to get started using it
- If you find a bug, report it to us. Without your help we can never hope to produce a bug free code.
- Help us to improve the documentation by providing information about documentation bugs
- Answer support requests in the **RMOL** discussion forums on SourceForge. If you know the answer to a question, help others to overcome their **RMOL** problems.
- Help us to improve our algorithms. If you know of a better way (e.g. that is faster or requires less memory) to implement some of our algorithms, then let us know.

- Help us to port [RMOL](#) to new platforms. If you manage to compile [RMOL](#) on a new platform, then tell us how you did it.
- Send us your code. If you have a good [RMOL](#) compatible code, which you can release under the LGPL, and you think it should be included in [RMOL](#), then send it to us.
- Become an [RMOL](#) developer. Send us an e-mail and tell what you can do for [RMOL](#).

8 Make a new release

8.1 Introduction

This document describes briefly the recommended procedure of releasing a new version of [RMOL](#) using a Linux development machine and the SourceForge project site.

The following steps are required to make a release of the distribution package.

- [Initialisation](#)
- [Release branch maintenance](#)
- [Commit and publish the release branch](#)
- [Create source packages \(tar-balls\)](#)
- [Upload the HTML documentation to SourceForge](#)
- [Generate the RPM packages](#)
- [Update distributed change log](#)
- [Create the binary package, including the documentation](#)
- [Upload the files to SourceForge](#)
- [Make a new post](#)
- [Send an email on the announcement mailing-list](#)

8.2 Initialisation

Clone locally the full [Git project](#):

```
cd ~
mkdir -p dev/sim
cd ~/dev/sim
git clone git://rmol.git.sourceforge.net/gitroot/rmol/rmol rmolgit
cd rmolgit
git checkout trunk
```

8.3 Release branch maintenance

Switch to the release branch, on your local clone, and merge the latest updates from the trunk. Decide about the new version to be released.

```
cd ~/dev/sim/rmolgit
git checkout releases
git merge trunk
```

Update the version in the various build system files, replacing the old version numbers by the correct ones:

```
vi CMakeLists.txt
vi autogen.sh
vi README
```

Update the version, add some news in the NEWS file, add a change-log in the ChangeLog file and in the RPM specification files:

```
vi NEWS
vi ChangeLog
vi rmol.spec
```

8.4 Commit and publish the release branch

Commit the new release:

```
cd ~/dev/sim/rmolgit
git add -A
git commit -m "[Release 0.5.0] Release of the 0.5.0 version of RMOL."
git push
```

8.5 Create source packages (tar-balls)

Create the distribution packages using the following command:

```
cd ~/dev/sim/rmolgit
git checkout releases
rm -rf build && mkdir -p build
cd build
export INSTALL_BASEDIR=/home/user/dev/deliveries
export LIBSUFFIX_4_CMAKE="-DLIB_SUFFIX=64"
cmake -DCMAKE_INSTALL_PREFIX=${INSTALL_BASEDIR}/rmol-0.5.0 \
  -DWITH_STDAIR_PREFIX=${INSTALL_BASEDIR}/stdair-stable \
  -DWITH_AIRAC_PREFIX=${INSTALL_BASEDIR}/airsched-stable \
  -DWITH_AIRAC_PREFIX=${INSTALL_BASEDIR}/airrac-stable \
  -DWITH_RMOL_PREFIX=${INSTALL_BASEDIR}/rmol-stable \
  -DWITH_RMOL_PREFIX=${INSTALL_BASEDIR}/airinv-stable \
  -DWITH_RMOL_PREFIX=${INSTALL_BASEDIR}/simfqt-stable \
  -DCMAKE_BUILD_TYPE:String=Debug -DINSTALL_DOC:BOOL=ON \
  ${LIBSUFFIX_4_CMAKE} ..
make check && make dist
make install
```

This will configure, compile and check the package. The output packages will be named, for instance, `rmol-0.5.0.tar.gz` and `rmol-0.5.0.tar.bz2`.

8.6 Upload the HTML documentation to SourceForge

In order to update the Web site files, either:

- **synchronise them with rsync and SSH:** Upload the just generated HTML (and PDF) documentation onto the **SourceForge Web site**.

```
cd ~/dev/sim/rmolgit/build
git checkout releases
rsync -aiv ${INSTALL_BASEDIR}/rmol-0.5.0/share/doc/rmol-0.5.0/html/ \
  your_sf_user,rmol@web.sourceforge.net:htdocs/
```

where `-aiv` options mean:

- `-a`: archive/mirror mode; equals `-rlptgoD` (no `-H`, `-A`, `-X`)
- `-v`: increase verbosity
- `-i`: output a change-summary for all updates

- Note the trailing slashes (/) at the end of both the source and target directories. It means that the content of the source directory (doc/html), rather than the directory itself, has to be copied into the content of the target directory.
- or use the [SourceForge Shell service](#).

8.7 Generate the RPM packages

Optionally, generate the RPM package (for instance, for [Fedora/RedHat](#)):

```
cd ~/dev/sim/rmolgit/build
git checkout releases
make dist
```

To perform this step, rpm-build, rpmlint and rpmdevtools have to be available on the system.

```
cp ../rmol.spec ~/dev/packages/SPECS \
  && cp rmol-0.5.0.tar.bz2 ~/dev/packages/SOURCES
cd ~/dev/packages/SPECS
rpmbuild -ba rmol.spec
cd ~/dev/packages
rpmlint -i SPECS/rmol.spec SRPMS/rmol-0.5.0-1.fc16.src.rpm \
  RPMS/noarch/rmol-* RPMS/i686/rmol-*
```

8.8 Update distributed change log

Update the NEWS and ChangeLog files with appropriate information, including what has changed since the previous release. Then commit and push the changes into the [RMOL's Git repository](#).

8.9 Create the binary package, including the documentation

Create the binary package, which includes HTML and PDF documentation, using the following command:

```
cd ~/dev/sim/rmolgit/build
git checkout releases
make package
```

The output binary package will be named, for instance, rmol-0.5.0-Linux.tar.bz2. That package contains both the HTML and PDF documentation. The binary package contains also the executables and shared libraries, as well as C++ header files, but all of those do not interest us for now.

8.10 Upload the files to SourceForge

Upload the distribution and documentation packages to the SourceForge server. Check [SourceForge help page on uploading software](#).

8.11 Make a new post

- submit a new entry in the [SourceForge project-related news feed](#)
- make a new post on the [SourceForge hosted WordPress blog](#)
- and update, if necessary, [Trac tickets](#).

8.12 Send an email on the announcement mailing-list

Finally, you should send an announcement to rmol-announce@lists.sourceforge.net (see <https://lists.sourceforge.net/lists/listinfo/rmol-announce> for the archives)

9 Installation

9.1 Table of Contents

- [Fedora/RedHat Linux distributions](#)
- [RMOL Requirements](#)
- [Basic Installation](#)
- [Compilers and Options](#)
- [Compiling For Multiple Architectures](#)
- [Installation Names](#)
- [Optional Features](#)
- [Particular systems](#)
- [Specifying the System Type](#)
- [Sharing Defaults](#)
- [Defining Variables](#)
- [‘cmake’ Invocation](#)

9.2 Fedora/RedHat Linux distributions

Note that on [Fedora/RedHat](#) Linux distributions, RPM packages are available and can be installed with your usual package manager. For instance:

```
yum -y install rmol-devel rmol-doc
```

RPM packages can also be available on the [SourceForge download site](#).

9.3 RMOL Requirements

[RMOL](#) should compile without errors or warnings on most GNU/Linux systems, on UNIX systems like Solaris SunOS, and on POSIX based environments for Microsoft Windows like Cygwin or MinGW with MSYS. It can be also built on Microsoft Windows NT/2000/XP/Vista/7 using Microsoft's Visual C++ .NET, but our support for this compiler is limited. For GNU/Linux, SunOS, Cygwin and MinGW we assume that you have at least the following GNU software installed on your computer:

- GNU Autotools:
 - [autoconf](#),
 - [automake](#),
 - [libtool](#),
 - [make](#), version 3.72.1 or later (check version with ``make --version``)
- [GCC](#) - GNU C++ Compiler (g++), version 4.3.x or later (check version with ``gcc --version``)
- [Boost](#) - C++ STL extensions, version 1.35 or later (check version with ``grep "define BOOST_LIB_VERSION" /usr/include/boost/version.hpp``)
- [MySQL](#) - Database client libraries, version 5.0 or later (check version with ``mysql --version``)
- [SOXI](#) - C++ database client library wrapper, version 3.0.0 or later (check version with ``soci-config --version``)

Optionally, you might need a few additional programs: [Doxygen](#), [LaTeX](#), [Dvips](#) and [Ghostscript](#), to generate the HTML and PDF documentation.

We strongly recommend that you use recent stable releases of the GCC, if possible. We do not actively work on supporting older versions of the GCC, and they may therefore (without prior notice) become unsupported in future releases of [RMOL](#).

9.4 Basic Installation

Briefly, the shell commands `./cmake .. && make install` should configure, build and install this package. The following more-detailed instructions are generic; see the `README` file for instructions specific to this package. Some packages provide this `INSTALL` file but do not implement all of the features documented below. The lack of an optional feature in a given package is not necessarily a bug. More recommendations for GNU packages can be found in the info page corresponding to "Makefile Conventions: (standards)Makefile Conventions".

The `cmake` shell script attempts to guess correct values for various system-dependent variables used during compilation. It uses those values to create a `Makefile` in each directory of the package. It may also create one or more `.h` files containing system-dependent definitions. Finally, it creates a `CMakeCache.txt` cache file that you can refer to in the future to recreate the current configuration, and files `CMakeFiles` containing compiler output (useful mainly for debugging `cmake`).

It can also use an optional file (typically called `config.cache` and enabled with `-cache-file=config.cache` or simply `-C`) that saves the results of its tests to speed up reconfiguring. Caching is disabled by default to prevent problems with accidental use of stale cache files.

If you need to do unusual things to compile the package, please try to figure out how `configure` could check whether to do them, and mail diffs or instructions to the address given in the `README` so they can be considered for the next release. If you are using the cache, and at some point `config.cache` contains results you don't want to keep, you may remove or edit it.

The file `CMakeLists.txt` is used to create the `Makefile` files.

The simplest way to compile this package is:

1. `cd` to the directory containing the package's source code and type `./cmake ..` to configure the package for your system. Running `cmake` is generally fast. While running, it prints some messages telling which features it is checking for.
2. Type `make` to compile the package.
3. Optionally, type `make check` to run any self-tests that come with the package, generally using the just-built uninstalled binaries.
4. Type `make install` to install the programs and any data files and documentation. When installing into a prefix owned by root, it is recommended that the package be configured and built as a regular user, and only the `make install` phase executed with root privileges.
5. You can remove the program binaries and object files from the source code directory by typing `make clean`. To also remove the files that `configure` created (so you can compile the package for a different kind of computer), type `make distclean`. There is also a `make maintainer-clean` target, but that is intended mainly for the package's developers. If you use it, you may have to get all sorts of other programs in order to regenerate files that came with the distribution.
6. Often, you can also type `make uninstall` to remove the installed files again. In practice, not all packages have tested that uninstallation works correctly, even though it is required by the GNU Coding Standards.

9.5 Compilers and Options

Some systems require unusual options for compilation or linking that the 'cmake' script does not know about. Run './cmake -help' for details on some of the pertinent environment variables.

You can give 'cmake' initial values for configuration parameters by setting variables in the command line or in the environment. Here is an example:

```
./cmake CC=c99 CFLAGS=-g LIBS=-lposix
```

See Also

[Defining Variables](#) for more details.

9.6 Compiling For Multiple Architectures

You can compile the package for more than one kind of computer at the same time, by placing the object files for each architecture in their own directory. To do this, you can use GNU 'make'. 'cd' to the directory where you want the object files and executables to go and run the 'configure' script. 'configure' automatically checks for the source code in the directory that 'configure' is in and in '..'. This is known as a "VPATH" build.

With a non-GNU 'make', it is safer to compile the package for one architecture at a time in the source code directory. After you have installed the package for one architecture, use 'make distclean' before reconfiguring for another architecture.

On MacOS X 10.5 and later systems, you can create libraries and executables that work on multiple system types-known as "fat" or "universal" binaries-by specifying multiple '-arch' options to the compiler but only a single '-arch' option to the preprocessor. Like this:

```
./configure CC="gcc -arch i386 -arch x86_64 -arch ppc -arch ppc64" \
           CXX="g++ -arch i386 -arch x86_64 -arch ppc -arch ppc64" \
           CPP="gcc -E" CXXCPP="g++ -E"
```

This is not guaranteed to produce working output in all cases, you may have to build one architecture at a time and combine the results using the 'lipo' tool if you have problems.

9.7 Installation Names

By default, 'make install' installs the package's commands under '/usr/local/bin', include files under '/usr/local/include', etc. You can specify an installation prefix other than '/usr/local' by giving 'configure' the option '-prefix=PREFIX', where PREFIX must be an absolute file name.

You can specify separate installation prefixes for architecture-specific files and architecture-independent files. If you pass the option '-exec-prefix=PREFIX' to 'configure', the package uses PREFIX as the prefix for installing programs and libraries. Documentation and other data files still use the regular prefix.

In addition, if you use an unusual directory layout you can give options like '-bindir=DIR' to specify different values for particular kinds of files. Run 'configure -help' for a list of the directories you can set and what kinds of files go in them. In general, the default for these options is

expressed in terms of `'${prefix}'`, so that specifying just `'-prefix'` will affect all of the other directory specifications that were not explicitly provided.

The most portable way to affect installation locations is to pass the correct locations to `'configure'`; however, many packages provide one or both of the following shortcuts of passing variable assignments to the `'make install'` command line to change installation locations without having to reconfigure or recompile.

The first method involves providing an override variable for each affected directory. For example, `'make install prefix=/alternate/directory'` will choose an alternate location for all directory configuration variables that were expressed in terms of `'${prefix}'`. Any directories that were specified during `'configure'`, but not in terms of `'${prefix}'`, must each be overridden at install time for the entire installation to be relocated. The approach of makefile variable overrides for each directory variable is required by the GNU Coding Standards, and ideally causes no recompilation. However, some platforms have known limitations with the semantics of shared libraries that end up requiring recompilation when using this method, particularly noticeable in packages that use GNU Libtool.

The second method involves providing the `'DESTDIR'` variable. For example, `'make install DESTDIR=/alternate/directory'` will prepend `'/alternate/directory'` before all installation names. The approach of `'DESTDIR'` overrides is not required by the GNU Coding Standards, and does not work on platforms that have drive letters. On the other hand, it does better at avoiding recompilation issues, and works well even when some directory options were not specified in terms of `'${prefix}'` at `'configure'` time.

9.8 Optional Features

If the package supports it, you can cause programs to be installed with an extra prefix or suffix on their names by giving `'cmake'` the option `'-program-prefix=PREFIX'` or `'-program-suffix=SUFFIX'`.

Some packages pay attention to `'-enable-FEATURE'` options to `'configure'`, where `FEATURE` indicates an optional part of the package. They may also pay attention to `'-with-PACKAGE'` options, where `PACKAGE` is something like `'gnu-as'` or `'x'` (for the X Window System). The `'README'` should mention any `'-enable-'` and `'-with-'` options that the package recognizes.

For packages that use the X Window System, `'configure'` can usually find the X include and library files automatically, but if it doesn't, you can use the `'configure'` options `'-x-includes=DIR'` and `'-x-libraries=DIR'` to specify their locations.

Some packages offer the ability to configure how verbose the execution of `'make'` will be. For these packages, running `'./configure -enable-silent-rules'` sets the default to minimal output, which can be overridden with `'make V=1'`; while running `'./configure -disable-silent-rules'` sets the default to verbose, which can be overridden with `'make V=0'`.

9.9 Particular systems

On HP-UX, the default C compiler is not ANSI C compatible. If GNU CC is not installed, it is recommended to use the following options in order to use an ANSI C compiler:

```
./configure CC="cc -Ae -D_XOPEN_SOURCE=500"
```

and if that doesn't work, install pre-built binaries of GCC for HP-UX.

On OSF/1 a.k.a. Tru64, some versions of the default C compiler cannot parse its '<wchar.h>' header file. The option '-nodtk' can be used as a workaround. If GNU CC is not installed, it is therefore recommended to try

```
./configure CC="cc"
```

and if that doesn't work, try

```
./configure CC="cc -nodtk"
```

On Solaris, don't put '/usr/ucb' early in your 'PATH'. This directory contains several dysfunctional programs; working variants of these programs are available in '/usr/bin'. So, if you need '/usr/ucb' in your 'PATH', put it *after* '/usr/bin'.

On Haiku, software installed for all users goes in '/boot/common', not '/usr/local'. It is recommended to use the following options:

```
./cmake -DCMAKE_INSTALL_PREFIX=/boot/common
```

9.10 Specifying the System Type

There may be some features 'configure' cannot figure out automatically, but needs to determine by the type of machine the package will run on. Usually, assuming the package is built to be run on the *same* architectures, 'configure' can figure that out, but if it prints a message saying it cannot guess the machine type, give it the '-build=TYPE' option. TYPE can either be a short name for the system type, such as 'sun4', or a canonical name which has the form CPU-COMPANY-SYSTEM

where SYSTEM can have one of these forms:

- OS
- KERNEL-OS

See the file 'config.sub' for the possible values of each field. If 'config.sub' isn't included in this package, then this package doesn't need to know the machine type.

If you are *building* compiler tools for cross-compiling, you should use the option '-target=TYPE' to select the type of system they will produce code for.

If you want to use a cross compiler, that generates code for a platform different from the build platform, you should specify the "host" platform (i.e., that on which the generated programs will eventually be run) with '-host=TYPE'.

9.11 Sharing Defaults

If you want to set default values for 'configure' scripts to share, you can create a site shell script called 'config.site' that gives default values for variables like 'CC', 'cache_file', and 'prefix'. 'configure' looks for 'PREFIX/share/config.site' if it exists, then 'PREFIX/etc/config.site' if it exists. Or, you can set the 'CONFIG_SITE' environment variable to the location of the site script. A warning: not all 'configure' scripts look for a site script.

9.12 Defining Variables

Variables not defined in a site shell script can be set in the environment passed to 'configure'. However, some packages may run configure again during the build, and the customized values of these variables may be lost. In order to avoid this problem, you should set them in the 'configure' command line, using 'VAR=value'. For example:

```
./configure CC=/usr/local2/bin/gcc
```

causes the specified 'gcc' to be used as the C compiler (unless it is overridden in the site shell script).

Unfortunately, this technique does not work for 'CONFIG_SHELL' due to an Autoconf bug. Until the bug is fixed you can use this workaround:

```
CONFIG_SHELL=/bin/bash /bin/bash ./configure CONFIG_SHELL=/bin/bash
```

9.13 'cmake' Invocation

'cmake' recognizes the following options to control how it operates.

- '-help', '-h' print a summary of all of the options to 'configure', and exit.
- '-help=short', '-help=recursive' print a summary of the options unique to this package's 'configure', and exit. The 'short' variant lists options used only in the top level, while the 'recursive' variant lists options also present in any nested packages.
- '-version', '-V' print the version of Autoconf used to generate the 'configure' script, and exit.
- '-cache-file=FILE' enable the cache: use and save the results of the tests in FILE, traditionally 'config.cache'. FILE defaults to '/dev/null' to disable caching.
- '-config-cache', '-C' alias for '-cache-file=config.cache'.
- '-quiet', '-silent', '-q' do not print messages saying which checks are being made. To suppress all normal output, redirect it to '/dev/null' (any error messages will still be shown).
- '-srcdir=DIR' look for the package's source code in directory DIR. Usually 'configure' can determine that directory automatically.
- '-prefix=DIR' use DIR as the installation prefix.

See Also

[Installation Names](#) for more details, including other options available for fine-tuning the installation locations.

- '-no-create', '-n' run the configure checks, but stop before creating any output files.

'cmake' also accepts some other, not widely useful, options. Run 'cmake -help' for more details.

The 'cmake' script produces an output like this:

```

cmake -DCMAKE_INSTALL_PREFIX=/home/user/dev/deliveries/rmol-99.99.99 -DLIB_SUFFIX=64 -DCMAKE_BUILD_TYPE:STRING
-- The C compiler identification is GNU
-- The CXX compiler identification is GNU
-- Check for working C compiler: /usr/lib64/ccache/gcc
-- Check for working C compiler: /usr/lib64/ccache/gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/lib64/ccache/c++
-- Check for working CXX compiler: /usr/lib64/ccache/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Requires Git without specifying any version
-- Current Git revision name: 56c6c98cf2cfb4008a0acd35d08075cf5f79e693 trunk
-- Requires Boost-1.41
-- Boost version: 1.46.0
-- Found the following Boost libraries:
--   program_options
--   date_time
--   iostreams
--   serialization
--   filesystem
--   unit_test_framework
--   python
-- Found Boost version: 1.46.0
-- Found BoostWrapper: /usr/include (Required is at least version "1.41")
-- Requires MySQL without specifying any version
-- Using mysql-config: /usr/bin/mysql_config
-- Found MySQL: /usr/lib64/mysql/libmysqlclient.so
-- Found MySQL version: 5.5.14
-- Requires SOCI-3.0
-- Using soci-config: /usr/bin/soci-config
-- SOCI headers are buried
-- Found SOCI: /usr/lib64/libsoci_core.so (Required is at least version "3.0")
-- Found SOCIMySQL: /usr/lib64/libsoci_mysql.so (Required is at least version "3.0")
-- Found SOCI with MySQL back-end support version: 3.0.0
-- Requires StdAir-0.35
-- Found StdAir version: 0.37.1
-- Requires Doxygen without specifying any version
-- Found Doxygen: /usr/bin/doxygen
-- Found DoxygenWrapper: /usr/bin/doxygen
-- Found Doxygen version: 1.7.4
-- Had to set the linker language for 'airraclib' to CXX
-- Had to set the linker language for 'rmollib' to CXX
-- Test 'UnconstrainerTest' to be built with 'UnconstrainerTestSuite.cpp'
-- Test 'ForecasterTest' to be built with 'ForecasterTestSuite.cpp'
-- Test 'OptimiseTest' to be built with 'OptimiseTestSuite.cpp'
-- Test 'BOMsForForecasterTest' to be built with 'bomsforforecaster.cpp'
--
-- =====
-- ---      Project Information      ---
-- ---
-- PROJECT_NAME .....: rmol
-- PACKAGE_PRETTY_NAME .....: RMOL
-- PACKAGE .....: rmol
-- PACKAGE_NAME .....: RMOL
-- PACKAGE_BRIEF .....: C++ library of Revenue Management and Optimisation classes and functions
-- PACKAGE_VERSION .....: 99.99.99
-- GENERIC_LIB_VERSION .....: 99.99.99
-- GENERIC_LIB_SOVERSION .....: 99.99
--
-- ---
-- ---      Build Configuration      ---
-- ---
-- Modules to build .....: airrac;rmol
-- Libraries to build/install .....: airraclib;rmollib
-- Binaries to build/install .....: airrac;rmol
-- Modules to test .....: rmol
-- Binaries to test .....: UnconstrainerTesttst;UnconstrainerTesttst;ForecasterTesttst;Unconstrained
--
-- * Module .....: airrac
-- + Layers to build .....: .;basic;bom;factory;command;service
-- + Dependencies on other layers :

```



```

-- + Libraries to build/install . : airraclib
-- + Executables to build/install : airrac
-- + Tests to perform ..... :
-- * Module ..... : rmol
-- + Layers to build ..... : .;basic;bom;factory;command;service
-- + Dependencies on other layers : airraclib
-- + Libraries to build/install . : rmolllib
-- + Executables to build/install : rmol
-- + Tests to perform ..... : UnconstrainerTesttst;UnconstrainerTesttst;ForecasterTesttst;Unconstrained
--
-- BUILD_SHARED_LIBS ..... : ON
-- CMAKE_BUILD_TYPE ..... : Debug
-- * CMAKE_C_FLAGS ..... :
-- * CMAKE_CXX_FLAGS ..... : -Wall -Werror
-- * BUILD_FLAGS ..... :
-- * COMPILE_FLAGS ..... :
-- CMAKE_MODULE_PATH ..... : /home/user/dev/sim/rmol/rmolgithub/config/
-- CMAKE_INSTALL_PREFIX ..... : /home/user/dev/deliveries/rmol-99.99.99
--
-- * Doxygen:
-- - DOXYGEN_VERSION ..... : 1.7.4
-- - DOXYGEN_EXECUTABLE ..... : /usr/bin/doxygen
-- - DOXYGEN_DOT_EXECUTABLE ..... : /usr/bin/dot
-- - DOXYGEN_DOT_PATH ..... : /usr/bin
--
-- -----
-- --- Installation Configuration ---
-- -----
-- INSTALL_LIB_DIR ..... : /home/user/dev/deliveries/rmol-99.99.99/lib64
-- INSTALL_BIN_DIR ..... : /home/user/dev/deliveries/rmol-99.99.99/bin
-- INSTALL_INCLUDE_DIR ..... : /home/user/dev/deliveries/rmol-99.99.99/include
-- INSTALL_DATA_DIR ..... : /home/user/dev/deliveries/rmol-99.99.99/share
-- INSTALL_SAMPLE_DIR ..... : /home/user/dev/deliveries/rmol-99.99.99/share/rmol/samples
-- INSTALL_DOC ..... : ON
--
-- -----
-- --- Packaging Configuration ---
-- -----
-- CPACK_PACKAGE_CONTACT ..... : Denis Arnaud <denis_arnaud - at - users dot sourceforge dot net>
-- CPACK_PACKAGE_VENDOR ..... : Denis Arnaud
-- CPACK_PACKAGE_VERSION ..... : 99.99.99
-- CPACK_PACKAGE_DESCRIPTION_FILE . : /home/user/dev/sim/rmol/rmolgithub/README
-- CPACK_RESOURCE_FILE_LICENSE .... : /home/user/dev/sim/rmol/rmolgithub/COPYING
-- CPACK_GENERATOR ..... : TBZ2
-- CPACK_DEBIAN_PACKAGE_DEPENDS ... :
-- CPACK_SOURCE_GENERATOR ..... : TBZ2;TGZ
-- CPACK_SOURCE_PACKAGE_FILE_NAME . : rmol-99.99.99
--
-- -----
-- --- External libraries ---
-- -----
--
-- * Boost:
-- - Boost_VERSION ..... : 104600
-- - Boost_LIB_VERSION ..... : 1_46
-- - Boost_HUMAN_VERSION ..... : 1.46.0
-- - Boost_INCLUDE_DIRS ..... : /usr/include
-- - Boost required components .. : program_options;date_time;iostreams;serialization;filesystem;unit_test_f
-- - Boost required libraries ... : optimized;/usr/lib64/libboost_iostreams-mt.so;debug;/usr/lib64/libboost_
--
-- * MySQL:
-- - MYSQL_VERSION ..... : 5.5.14
-- - MYSQL_INCLUDE_DIR ..... : /usr/include/mysql
-- - MYSQL_LIBRARIES ..... : /usr/lib64/mysql/libmysqlclient.so
--
-- * SOCI:
-- - SOCI_VERSION ..... : 3.0.0
-- - SOCI_INCLUDE_DIR ..... : /usr/include/soci
-- - SOCI_MYSQL_INCLUDE_DIR ..... : /usr/include/soci
-- - SOCI_LIBRARIES ..... : /usr/lib64/libsoci_core.so
-- - SOCI_MYSQL_LIBRARIES ..... : /usr/lib64/libsoci_mysql.so
--
-- * StdAir:

```

```
-- - STDAIR_VERSION ..... : 0.37.1
-- - STDAIR_BINARY_DIRS ..... : /home/user/dev/deliveries/stdair-0.37.1/bin
-- - STDAIR_EXECUTABLES ..... : stdair
-- - STDAIR_LIBRARY_DIRS ..... : /home/user/dev/deliveries/stdair-0.37.1/lib64
-- - STDAIR_LIBRARIES ..... : stdairlib;stdairuiclib
-- - STDAIR_INCLUDE_DIRS ..... : /home/user/dev/deliveries/stdair-0.37.1/include
-- - STDAIR_SAMPLE_DIR ..... : /home/user/dev/deliveries/stdair-0.37.1/share/stdair/samples
--
-- Change a value with: cmake -D<Variable>=<Value>
-- =====
--
-- Configuring done
-- Generating done
-- Build files have been written to: /home/user/dev/sim/rmol/rmolgithub/build
```

It is recommended that you check if your library has been compiled and linked properly and works as expected. To do so, you should execute the testing process 'make check'. As a result, you should obtain a similar report:

```
[ 0%] Built target hdr_cfg_rmol
[ 0%] Built target hdr_cfg_airrac
[ 30%] Built target airraclib
[ 86%] Built target rmolib
[ 90%] Built target BOMsForForecasterTesttst
[ 93%] Built target UnconstrainerTesttst
[ 96%] Built target ForecasterTesttst
[100%] Built target OptimiseTesttst
Scanning dependencies of target check_rmoltst
Test project /home/user/dev/sim/rmol/rmolgithub/build/test/rmol
  Start 1: UnconstrainerTesttst
1/4 Test #1: UnconstrainerTesttst ..... Passed    0.04 sec
  Start 2: ForecasterTesttst
2/4 Test #2: ForecasterTesttst ..... Passed    0.04 sec
  Start 3: OptimiseTesttst
3/4 Test #3: OptimiseTesttst ..... Passed    0.44 sec
  Start 4: BOMsForForecasterTesttst
4/4 Test #4: BOMsForForecasterTesttst ..... Passed    0.02 sec

100% tests passed, 0 tests failed out of 4

Total Test time (real) = 0.78 sec
[100%] Built target check_rmoltst
Scanning dependencies of target check
[100%] Built target check
```

Check if all the executed tests PASSED. If not, please contact us by filling a [bug-report](#).

Finally, you should install the compiled and linked library, include files and (optionally) HTML and PDF documentation by typing:

```
make install
```

Depending on the PREFIX settings during configuration, you might need the root (administrator) access to perform this step.

Eventually, you might invoke the following command

```
make clean
```

to remove all files created during compilation process, or even

```
cd ~/dev/sim/rmolgit
rm -rf build && mkdir build
cd build
```

to remove everything.

10 Linking with RMOL

10.1 Table of Contents

- [Introduction](#)
- [Using the pkg-config command](#)
- [Using the rmol-config script](#)
- [M4 macro for the GNU Autotools](#)
- [Using RMOL with dynamic linking](#)

10.2 Introduction

There are two convenient methods of linking your programs with the [RMOL](#) library. The first one employs the `'pkg-config'` command (see <http://pkgconfig.freedesktop.org/>), whereas the second one uses `'rmol-config'` script. These methods are shortly described below.

10.3 Using the pkg-config command

`'pkg-config'` is a helper tool used when compiling applications and libraries. It helps you insert the correct compiler and linker options. The syntax of the `'pkg-config'` is as follows:

```
pkg-config <options> <library_name>
```

For instance, assuming that you need to compile an [RMOL](#) based program `'my_prog.cpp'`, you should use the following command:

```
g++ `pkg-config --cflags rmol` -o my_prog my_prog.cpp `pkg-config --libs rmol`
```

For more information see the `'pkg-config'` man pages.

10.4 Using the rmol-config script

[RMOL](#) provides a shell script called `rmol-config`, which is installed by default in `'$prefix/bin'` (`'/usr/local/bin'`) directory. It can be used to simplify compilation and linking of [RMOL](#) based programs. The usage of this script is quite similar to the usage of the `'pkg-config'` command.

Assuming that you need to compile the program `'my_prog.cpp'` you can now do that with the following command:

```
g++ `rmol-config --cflags` -o my_prog_opt my_prog.cpp `rmol-config --libs`
```

A list of `'rmol-config'` options can be obtained by typing:

```
rmol-config --help
```

If the `'rmol-config'` command is not found by your shell, you should add its location `'$prefix/bin'` to the `PATH` environment variable, e.g.:

```
export PATH=/usr/local/bin:$PATH
```

10.5 M4 macro for the GNU Autotools

A M4 macro file is delivered with [RMOL](#), namely 'rmol.m4', which can be found in, e.g., '/usr/share/aclocal'. When used by a 'configure' script, thanks to the 'AM_PATH_RMOL' macro (specified in the M4 macro file), the following Makefile variables are then defined:

- 'RMOL_VERSION' (e.g., defined to 0.23.0)
- 'RMOL_CFLAGS' (e.g., defined to '-I\${prefix}/include')
- 'RMOL_LIBS' (e.g., defined to '-L\${prefix}/lib -lrmol')

10.6 Using RMOL with dynamic linking

When using static linking some of the library routines in [RMOL](#) are copied into your executable program. This can lead to unnecessary large executables. To avoid having too large executable files you may use dynamic linking instead. Dynamic linking means that the actual linking is performed when the program is executed. This requires that the system is able to locate the shared [RMOL](#) library file during your program execution. If you install the [RMOL](#) library using a non-standard prefix, the 'LD_LIBRARY_PATH' environment variable might be used to inform the linker of the dynamic library location, e.g.:

```
export LD_LIBRARY_PATH=<RMOL installation prefix>/lib:$LD_LIBRARY_PATH
```

11 Test Rules

This section describes rules how the functionality of the IT++ library should be verified. In the 'tests' subdirectory test files are provided. All functionality should be tested using these test files.

11.1 The Test File

Each new IT++ module/class should be accompanied with a test file. The test file is an implementation in C++ that tests the functionality of a function/class or a group of functions/classes called modules. The test file should test relevant parameter settings and input/output relations to guarantee correct functionality of the corresponding classes/functions. The test files should be maintained using version control and updated whenever new functionality is added to the IT++ library.

The test file should print relevant data to a standard output that can be used to verify the functionality. All relevant parameter settings should be tested.

The test file should be placed in the 'tests' subdirectory and should have a name ending with '_test.cpp'.

11.2 The Reference File

Consider a test file named 'module_test.cpp'. A reference file named 'module_test.ref' should accompany the test file. The reference file contains a reference printout of the standard output generated when running the test program. The reference file should be maintained using version control and updated according to the test file.

11.3 Testing IT++ Library

One can compile and execute all test programs from 'tests' subdirectory by typing

```
% make check
```

after successful compilation of the IT++ library.

12 Users Guide

12.1 Table of Contents

- [Introduction](#)
- [Get Started](#)
 - [Get the RMOL library](#)
 - [Build the RMOL project](#)
 - [Build and Run the Tests](#)
 - [Install the RMOL Project \(Binaries, Documentation\)](#)
- [Exploring the Predefined BOM Tree](#)
 - [Forecaster BOM Tree](#)
 - [Optimiser BOM Tree](#)
- [Extending the BOM Tree](#)

12.2 Introduction

The [RMOL](#) library contains classes for revenue management. This document does not cover all the aspects of the [RMOL](#) library. It does however explain the most important things you need to know in order to start using [RMOL](#).

12.3 Get Started

12.3.1 Get the RMOL library

12.3.2 Build the RMOL project

To run the configuration script the first time, go to the top directory (where the [RMOL](#) package has been un-packed), and issue the following command:

- `mkdir -p build && cd build && cmake ..`
- `make`

Note

The [RMOL](#) project can either be cloned from the [Git Repository](#) or downloaded as a tar-ball package from the [Sourceforge Web site](#).

12.3.3 Build and Run the Tests

12.3.4 Install the RMOL Project (Binaries, Documentation)

12.4 Exploring the Predefined BOM Tree

[RMOL](#) predefines a BOM (Business Object Model) tree specific to the airline IT arena.

12.4.1 Forecaster BOM Tree

- [RMOL::EMDetruncator](#)
- [RMOL::Detruncator](#)
- [RMOL::Forecaster](#)

12.4.2 Optimiser BOM Tree

- [RMOL::DPOptimiser](#)
- [RMOL::MCOptimiser](#)
- [RMOL::Optimiser](#)

12.5 Extending the BOM Tree

13 Supported Systems

13.1 Table of Contents

- [Introduction](#)
- [RMOL 0.23.x](#)
 - [Linux Systems](#)
 - * [Fedora Core 4 with ATLAS](#)
 - * [Gentoo Linux with ACML](#)
 - * [Gentoo Linux with ATLAS](#)
 - * [Gentoo Linux with MKL](#)
 - * [Gentoo Linux with NetLib's BLAS and LAPACK](#)
 - * [Red Hat Enterprise Linux with RMOL External](#)
 - * [SUSE Linux 10.0 with NetLib's BLAS and LAPACK](#)
 - * [SUSE Linux 10.0 with MKL](#)
 - [Windows Systems](#)
 - * [Microsoft Windows XP with Cygwin](#)
 - * [Microsoft Windows XP with Cygwin and ATLAS](#)
 - * [Microsoft Windows XP with Cygwin and ACML](#)
 - * [Microsoft Windows XP with MinGW, MSYS and ACML](#)
 - * [Microsoft Windows XP with MinGW, MSYS and RMOL External](#)
 - * [Microsoft Windows XP with MS Visual C++ and Intel MKL](#)
 - [Unix Systems](#)
 - * [SunOS 5.9 with RMOL External](#)
- [RMOL 3.9.1](#)
- [RMOL 3.9.0](#)
- [RMOL 3.8.1](#)

13.2 Introduction

This page is intended to provide a list of [RMOL](#) supported systems, i.e. the systems on which configuration, installation and testing process of the [RMOL](#) library has been successful. Results are grouped based on minor release number. Therefore, only the latest tests for bug-fix releases are included. Besides, the information on this page is divided into sections dependent on the operating system.

Where necessary, some extra information is given for each tested configuration, e.g. external libraries installed, configuration commands used, etc.

If you manage to compile, install and test the [RMOL](#) library on a system not mentioned below, please let us know, so we could update this database.

13.3 RMOL 0.23.x

13.3.1 Linux Systems

13.3.1.1 Fedora Core 4 with ATLAS

- **Platform:** Intel Pentium 4
- **Operating System:** Fedora Core 4 (x86)
- **Compiler:** g++ (GCC) 4.0.2 20051125
- **RMOL release:** 0.23.0
- **External Libraries:** From FC4 distribution:
 - `fftw3.i386-3.0.1-3`
 - `fftw3-devel.i386-3.0.1-3`
 - `atlas-sse2.i386-3.6.0-8.fc4`
 - `atlas-sse2-devel.i386-3.6.0-8.fc4`
 - `blas.i386-3.0-35.fc4`
 - `lapack.i386-3.0-35.fc4`
- **Tests Status:** All tests PASSED
- **Comments:** RMOL configured with:


```
% CXXFLAGS="-O3 -pipe -march=pentium4" ./configure
```
- **Date:** March 7, 2006
- **Tester:** Tony Ottosson

13.3.1.2 Gentoo Linux with ACML

- **Platform:** AMD Sempron 3000+
- **Operating System:** Gentoo Linux 2006.0 (x86 arch)
- **Compiler(s):** g++ (GCC) 3.4.5
- **RMOL release:** 0.23.1
- **External Libraries:** Compiled and installed from portage tree:
 - `sci-libs/acml-3.0.0`
- **Tests Status:** All tests PASSED
- **Comments:** BLAS and LAPACK libs set by using the following system commands:


```
% eselect blas set ACML
% eselect lapack set ACML
```

RMOL configured with:

```
% export CPPFLAGS="-I/usr/include/acml"
% ./configure --with-blas="-lblas"
```
- **Date:** March 31, 2006
- **Tester:** Adam Piatyszek (ediap)

13.3.1.3 Gentoo Linux with ATLAS

- **Platform:** Intel Pentium M Centrino
- **Operating System:** Gentoo Linux 2006.0 (x86)
- **Compiler:** g++ (GCC) 3.4.5
- **RMOL release:** 0.23.1
- **External Libraries:** Compiled and installed from portage tree:
 - sci-libs/fftw-3.1
 - sci-libs/blas-atlas-3.6.0-r1
 - sci-libs/lapack-atlas-3.6.0
- **Tests Status:** All tests PASSED
- **Comments:** BLAS and LAPACK libs set by using the following system commands:

```
% eselect blas set ATLAS
% eselect lapack set ATLAS
```

RMOL configured with:

```
% ./configure --with-blas="-lblas"
```

- **Date:** March 31, 2006
- **Tester:** Adam Piatyszek (ediap)

13.3.1.4 Gentoo Linux with MKL

- **Platform:** Intel Pentium M Centrino
- **Operating System:** Gentoo Linux 2006.0 (x86 arch)
- **Compiler:** g++ (GCC) 3.4.5
- **RMOL release:** 0.23.0
- **External Libraries:** Intel Math Kernel Library (MKL) 8.0.1 installed manually in the following directory:
/opt/intel/mkl/8.0.1
- **Tests Status:** All tests PASSED
- **Comments:** RMOL configured using the following commands:

```
% export LDFLAGS="-L/opt/intel/mkl/8.0.1/lib/32"
% export CPPFLAGS="-I/opt/intel/mkl/8.0.1/include"
% ./configure
```

- **Date:** February 28, 2006
- **Tester:** Adam Piatyszek (ediap)

13.3.1.5 Gentoo Linux with NetLib's BLAS and LAPACK

- **Platform:** Intel Pentium M Centrino
- **Operating System:** Gentoo Linux 2006.0 (x86)
- **Compiler:** g++ (GCC) 3.4.5
- **RMOL release:** 0.23.1
- **External Libraries:** Compiled and installed from portage tree:
 - sci-libs/fftw-3.1
 - sci-libs/blas-reference-19940131-r2
 - sci-libs/cblas-reference-20030223
 - sci-libs/lapack-reference-3.0-r2
- **Tests Status:** All tests PASSED
- **Comments:** BLAS and LAPACK libs set by using the following system commands:

```
% blas-config reference
% lapack-config reference
```

RMOL configured with:

```
% ./configure --with-blas="-lblas"
```

- **Date:** March 31, 2006
- **Tester:** Adam Piatyszek (ediap)

13.3.1.6 Red Hat Enterprise Linux with RMOL External

- **Platform:** Intel Pentium 4
- **Operating System:** Red Hat Enterprise Linux AS release 4 (Nahant Update 2)
- **Compiler:** g++ (GCC) 3.4.4 20050721 (Red Hat 3.4.4-2)
- **RMOL release:** 0.23.0
- **External Libraries:** BLAS, CBLAS, LAPACK and FFTW libraries from [RMOL External 2.1.1](#) package
- **Tests Status:** All tests PASSED
- **Date:** March 7, 2006
- **Tester:** Erik G. Larsson

13.3.1.7 SUSE Linux 10.0 with NetLib's BLAS and LAPACK

- **Platform:** Intel Pentium 4 CPU 3.20GHz (64-bit)
- **Operating System:** SUSE Linux 10.0 (x86_64)
- **Compiler(s):** g++ (GCC) 4.0.2
- **RMOL release:** 0.23.0
- **External Libraries:** BLAS, LAPACK and FFTW libraries installed from OpenSuse 10.0 RPM repository:
 - blas-3.0-926
 - lapack-3.0-926
 - fftw3-3.0.1-114

- fftw3-threads-3.0.1-114
- fftw3-devel-3.0.1-114

- **Tests Status:** All tests PASSED

- **Comments:** [RMOL](#) configured with:

```
% export CXXFLAGS="-m64 -march=nocona -O3 -pipe"
% ./configure --with-lapack="/usr/lib64/liblapack.so.3"
```

- **Date:** March 1, 2006
- **Tester:** Adam Piatyszek (ediap)

13.3.1.8 SUSE Linux 10.0 with MKL

- **Platform:** Intel Pentium 4 CPU 3.20GHz (64-bit)
- **Operating System:** SUSE Linux 10.0 (x86_64)
- **Compiler(s):** g++ (GCC) 4.0.2
- **[RMOL](#) release:** 0.23.0
- **External Libraries:** Intel Math Kernel Library (MKL) 8.0.1 installed manually in the following directory:
/opt/intel/mkl/8.0.1
- **Tests Status:** All tests PASSED
- **Comments:** [RMOL](#) configured with:

```
% export CXXFLAGS="-m64 -march=nocona -O3 -pipe"
% export LDFLAGS="-L/opt/intel/mkl/8.0.1/lib/em64t"
% export CPPFLAGS="-I/opt/intel/mkl/8.0.1/include"
% ./configure
```

- **Date:** March 1, 2006
- **Tester:** Adam Piatyszek (ediap)

13.3.2 Windows Systems

13.3.2.1 Microsoft Windows XP with Cygwin

- **Platform:** AMD Sempron 3000+
- **Operating System:** Microsoft Windows XP SP2, Cygwin 1.5.19-4
- **Compiler(s):** g++ (GCC) 3.4.4 (cygming special)
- **[RMOL](#) release:** 0.23.1
- **External Libraries:** Installed from Cygwin's repository:

- fftw-3.0.1-2
- fftw-dev-3.0.1-1
- lapack-3.0-4

- **Tests Status:** All tests PASSED
- **Comments:** Only static library can be built. [RMOL](#) configured with:

```
% ./configure
```

- **Date:** March 31, 2006
- **Tester:** Adam Piatyszek (ediap)

13.3.2.2 Microsoft Windows XP with Cygwin and ATLAS

- **Platform:** AMD Sempron 3000+
- **Operating System:** Microsoft Windows XP SP2, Cygwin 1.5.19-4
- **Compiler(s):** g++ (GCC) 3.4.4 (cygming special)
- **RMOL release:** 0.23.1
- **External Libraries:** Installed from Cygwin's repository:
 - fftw-3.0.1-2
 - fftw-dev-3.0.1-1

ATLAS BLAS and LAPACK libraries from [RMOL](#) External 2.1.1 package configured using:

```
% ./configure --enable-atlas --disable-fftw
```

- **Tests Status:** All tests PASSED
- **Comments:** Only static library can be built. [RMOL](#) configured with:

```
% export LDFLAGS="-L/usr/local/lib"
% ./configure
```

- **Date:** March 31, 2006
- **Tester:** Adam Piatyszek (ediap)

13.3.2.3 Microsoft Windows XP with Cygwin and ACML

- **Platform:** AMD Sempron 3000+
- **Operating System:** Microsoft Windows XP SP2, Cygwin 1.5.19-4
- **Compiler(s):** g++ (GCC) 3.4.4 (cygming special)
- **RMOL release:** 0.23.2
- **External Libraries:** ACML version 3.1.0 (acml3.1.0-32-win32-g77.exe) installed into a default directory, i.e. "c:\Program Files\AMD\acml3.1.0"
- **Tests Status:** All tests PASSED
- **Comments:** Only static library can be built. [RMOL](#) configured with:

```
% export LDFLAGS="-L/cygdrive/c/Progra~1/AMD/acml3.1.0/gnu32/lib"
% export CPPFLAGS="-I/cygdrive/c/Progra~1/AMD/acml3.1.0/gnu32/include"
% ./configure --enable-debug
```

- **Date:** May 15, 2006
- **Tester:** Adam Piatyszek (ediap)

13.3.2.4 Microsoft Windows XP with MinGW, MSYS and ACML

- **Platform:** AMD Sempron 3000+
- **Operating System:** Microsoft Windows XP SP2, MinGW 5.0.2, MSYS 1.0.10
- **Compiler(s):** g++ (GCC) 3.4.4 (mingw special)
- **RMOL release:** 0.23.2
- **External Libraries:** ACML version 3.1.0 (acml3.1.0-32-win32-g77.exe) installed into a default directory, i.e. "c:\Program Files\AMD\acml3.1.0"
- **Tests Status:** All tests PASSED
- **Comments:** Only static library can be built. [RMOL](#) configured with:

```
% export LDFLAGS="-L/c/Program Files/AMD/acml3.1.0/gnu32/lib"
% export CPPFLAGS="-I/c/Program Files/AMD/acml3.1.0/gnu32/include"
% ./configure --enable-debug
```

- **Date:** May 15, 2006
- **Tester:** Adam Piatyszek (ediap)

13.3.2.5 Microsoft Windows XP with MinGW, MSYS and RMOL External

- **Platform:** AMD Sempron 3000+
- **Operating System:** Microsoft Windows XP SP2, MinGW 5.0.2, MSYS 1.0.10
- **Compiler(s):** g++ (GCC) 3.4.4 (mingw special)
- **RMOL release:** 0.23.5
- **External Libraries:** BLAS, CBLAS, LAPACK and FFTW libraries from [RMOL External 2.2.0](#) package
- **Tests Status:** All tests PASSED
- **Comments:** Only static library can be built. [RMOL](#) configured with:

```
% export LDFLAGS="-L/usr/local/lib"
% export CPPFLAGS="-I/usr/local/include"
% export CXXFLAGS="-Wall -O3 -march=athlon-tbird -pipe"
% ./configure --disable-html-doc
```

- **Date:** August 11, 2006
- **Tester:** Adam Piatyszek (ediap)

13.3.2.6 Microsoft Windows XP with MS Visual C++ and Intel MKL

- **Platform:** AMD Sempron 3000+
- **Operating System:** Microsoft Windows XP SP2
- **Compiler(s):** Microsoft Visual C++ 2005 .NET
- **RMOL release:** 0.23.5
- **External Libraries:** Intel Math Kernel Library (MKL) 8.1 installed manually in the following directory: "C:\Program Files\Intel\MKL\8.1"
- **Tests Status:** Not fully tested. Some [RMOL](#) based programs compiled and run with success.
- **Comments:** Only static library can be built. [RMOL](#) built by opening the "win32\rmol.vcproj" project file in MSVC++ and executing "Build -> Build Solution" command from menu.
- **Date:** August 11, 2006
- **Tester:** Adam Piatyszek (ediap)

13.3.3 Unix Systems

13.3.3.1 SunOS 5.9 with RMOL External

- **Platform:** SUNW, Sun-Blade-100 (SPARC)
- **Operating System:** SunOS 5.9 Generic_112233-10
- **Compiler(s):** g++ (GCC) 3.4.5
- **RMOL release:** 0.23.2
- **External Libraries:** BLAS, CBLAS, LAPACK and FFTW libraries from [RMOL](#) External 2.1.1 package. The following configuration command has been used:

```
% export CFLAGS="-mcpu=ultrasparc -O2 -pipe -funroll-all-loops"
% ./configure
```

- **Tests Status:** All tests PASSED
- **Comments:** [RMOL](#) configured with:

```
% export LDFLAGS="-L/usr/local/lib"
% export CPPFLAGS="-I/usr/local/include"
% export CXXFLAGS="-mcpu=ultrasparc -O2 -pipe"
% ./configure --enable-debug
```

- **Date:** May 15, 2006
- **Tester:** Adam Piatyszek (ediap)

14 RMOL Supported Systems (Previous Releases)

14.1 RMOL 3.9.1

14.2 RMOL 3.9.0

14.3 RMOL 3.8.1

15 Tutorials

15.1 Table of Contents

- [Introduction](#)
 - [Preparing the StdAir Project for Development](#)
- [Build a Predefined BOM Tree](#)
 - [Instantiate the BOM Root Object](#)
 - [Instantiate the \(Airline\) Inventory Object](#)
 - [Link the Inventory Object with the BOM Root](#)
 - [Build Another Airline Inventory](#)
 - [Dump The BOM Tree Content](#)
 - [Result of the Tutorial Program](#)
- [Extend the Pre-Defined BOM Tree](#)
 - [Extend an Airline Inventory Object](#)
 - [Build the Specific BOM Objects](#)
 - [Result of the Tutorial Program](#)

15.2 Introduction

This page contains some tutorial examples that will help you getting started using StdAir. Most examples show how to construct some simple business objects, i.e., instances of the so-named Business Object Model (BOM).

15.2.1 Preparing the StdAir Project for Development

The source code for these examples can be found in the `batches` and `test/stdair` directories. They are compiled along with the rest of the `StdAir` project. See the User Guide ([Users Guide](#)) for more details on how to build the `StdAir` project.

15.3 Build a Predefined BOM Tree

A few steps:

- [Instantiate the BOM Root Object](#)
- [Instantiate the \(Airline\) Inventory Object](#)
- [Link the Inventory Object with the BOM Root](#)

15.3.1 Instantiate the BOM Root Object

First, a BOM root object (i.e., a root for all the classes in the project) is instantiated by the `stdair::STD-AIR_ServiceContext` context object, when the `stdair::STDAIR_Service` is itself instantiated. The corresponding `StdAir` type (class) is `stdair::BomRoot`.

In the following sample, that object is named `ioBomRoot`, and is given as input/output parameter of the `stdair::CmdBomManager::buildSampleBom()` method:

15.3.2 Instantiate the (Airline) Inventory Object

An airline inventory object can then be instantiated. Let us give it the "BA" airline code (corresponding to [British Airways](#)) as the object key. That is, an object (let us name it `lBAKey`) of type (class) `stdair::InventoryKey` has first to be instantiated.

Thanks to that key, an airline inventory object, i.e. of type (class) `stdair::Inventory`, can be instantiated. Let us name that airline inventory object `lBAInv`.

15.3.3 Link the Inventory Object with the BOM Root

Then, both objects have to be linked: the airline inventory object (`stdair::Inventory`) has to be linked with the root of the BOM tree (`stdair::BomRoot`). That operation is as simple as using the `stdair::FacBomManager::addToListAndMap()` method:

15.3.4 Build Another Airline Inventory

Another airline inventory object, corresponding to the Air France (**Air France**) company, is instantiated the same way:

See the corresponding full program (cmd_bom_manager_cpp) for more details.

15.3.5 Dump The BOM Tree Content

From the BomRoot (of type stdair : : BomRoot) object instance, the list of airline inventories (of type stdair : : Inventory) can then be retrieved...

... and browsed:

See the corresponding full program (bom_display_cpp) for more details.

15.3.6 Result of the Tutorial Program

When the stdair.cpp program is run (with the -b option), the output should look like:

```
[D]../batches/stdair.cpp:243: Welcome to stdair
[D]../batches/stdair/command/CmdBomManager.cpp:41: StdAir will build the BOM tree
    from built-in specifications.
[D]../batches/stdair.cpp:286:
=====
BomRoot:  -- ROOT  --
=====
+++++
Inventory: BA
+++++
*****
FlightDate: BA9, 2011-Jun-10
*****
*****
Leg-Dates:
-----
Flight, Leg, BoardDate, BoardTime, OffDate, OffTime, Date Offset, Time Offset,
    Elapsed, Distance, Capacity,
BA9 2011-Jun-10, LHR-BKK, 2011-Jun-10, 21:45:00, 2011-Jun-11, 15:40:00, 11:05:
    00, 1, 06:50:00, 9900, 0,
BA9 2011-Jun-10, BKK-SYD, 2011-Jun-11, 17:05:00, 2011-Jun-12, 15:40:00, 09:05:
    00, 1, 13:30:00, 8100, 0,
*****
*****
LegCabins:
-----
Flight, Leg, Cabin, OffedCAP, PhycAP, RgdADJ, AU, UPR, SS, Staff, WL, Group,
    CommSpace, AvPool, Av1, NAV, GAV, ACP, ETB, BidPrice,
BA9 2011-Jun-10, LHR-BKK 2011-Jun-10, Y, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9, 9, 0,
    0, 3.52965e-319, 0, 0,
BA9 2011-Jun-10, BKK-SYD 2011-Jun-11, Y, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9, 9, 0,
    0, 0, 0, 0,
*****
*****
Buckets:
-----
Flight, Leg, Cabin, Yield, AU/SI, SS, AV,
*****
*****
SegmentCabins:
-----
Flight, Segment, Cabin, FF, Bkgs, MIN, UPR, CommSpace, AvPool, BP,
BA9 2011-Jun-10, LHR-SYD 2011-Jun-10, Y, EcoSaver, 0, 0, 0, 0, 9, 0,
BA9 2011-Jun-10, LHR-BKK 2011-Jun-10, Y, EcoSaver, 0, 0, 0, 0, 9, 0,
BA9 2011-Jun-10, BKK-SYD 2011-Jun-11, Y, EcoSaver, 0, 0, 0, 0, 9, 0,
```

```

*****
*****
Subclasses:
-----
Flight, Segment, Cabin, FF, Subclass, MIN/AU (Prot), Nego, NS%, OB%, Bkgs,
    GrpBks (pdg), StfBkgs, WLBkgs, ETB, ClassAvl, RevAvl, SegAvl,
BA9 2011-Jun-10, LHR-SYD 2011-Jun-10, Y, EcoSaver, Q, 0 (0), 0, 0, 0, 0, 0 (0),
    0, 0, 0, 0, 0, 0,
+++++
Inventory: AF
+++++
*****
FlightDate: AF84, 2011-Mar-20
*****
*****
Leg-Dates:
-----
Flight, Leg, BoardDate, BoardTime, OffDate, OffTime, Date Offset, Time Offset,
    Elapsed, Distance, Capacity,
AF84 2011-Mar-20, CDG-SFO, 2011-Mar-20, 10:40:00, 2011-Mar-20, 12:50:00, 11:10:
    00, 0, -09:00:00, 9900, 0,
*****
*****
LegCabins:
-----
Flight, Leg, Cabin, OffedCAP, PhycAP, RgdADJ, AU, UPR, SS, Staff, WL, Group,
    CommSpace, AvPool, Avl, NAV, GAV, ACP, ETB, BidPrice,
AF84 2011-Mar-20, CDG-SFO 2011-Mar-20, Y, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9, 9, 0
    , 0, 0, 0, 0,
*****
*****
Buckets:
-----
Flight, Leg, Cabin, Yield, AU/SI, SS, AV,
*****
*****
SegmentCabins:
-----
Flight, Segment, Cabin, FF, Bkgs, MIN, UPR, CommSpace, AvPool, BP,
AF84 2011-Mar-20, CDG-SFO 2011-Mar-20, Y, EcoSaver, 0, 0, 0, 0, 9, 0,
*****
*****
Subclasses:
-----
Flight, Segment, Cabin, FF, Subclass, MIN/AU (Prot), Nego, NS%, OB%, Bkgs,
    GrpBks (pdg), StfBkgs, WLBkgs, ETB, ClassAvl, RevAvl, SegAvl,
AF84 2011-Mar-20, CDG-SFO 2011-Mar-20, Y, EcoSaver, Q, 0 (0), 0, 0, 0, 0, 0 (0)
    , 0, 0, 0, 0, 0, 0,

```

See the corresponding full program (batch_stdair_cpp) for more details.

15.4 Extend the Pre-Defined BOM Tree

Now that we master how to instantiate the pre-defined StdAir classes, let us see how to extend that BOM.

15.4.1 Extend an Airline Inventory Object

For instance, let us assume that some (IT) provider (e.g., you) would like to have a specific implementation of the `Inventory` object. The corresponding class has just to extend the `stdair::Inventory` class:

The STL containers have to be defined accordingly too:

See the full class definition (test_archi_inv_hpp) and implementation (test_archi_inv_cpp) for more details.

15.4.2 Build the Specific BOM Objects

The BOM root object (`stdair::BomRoot`) is instantiated the classical way:

Then, the specific implementation of the airline inventory object (`myprovider::Inventory`) can be instantiated the same way as a standard `Inventory` (`stdair::Inventory`) would be:

Then, the specific implementation of the airline inventory object (`myprovider::Inventory`) is linked to the root of the BOM tree (`stdair::BomRoot`) the same way as the standard `Inventory` (`stdair::Inventory`) would be:

Another specific airline inventory object is instantiated the same way:

From the `BomRoot` (of type `stdair::BomRoot`) object instance, the list of specific airline inventories (of type `stdair::Inventory`) can then be retrieved...

... and browsed:

15.4.3 Result of the Tutorial Program

When this program is run, the output should look like:

```
Inventory: BA
Inventory: AF
```

See the corresponding full program (`StandardAirlineITTestSuite_cpp`) for more details.

16 Command-Line Test to Demonstrate How To Test the RMOL Project

```
*/
// //////////////////////////////////////
// Import section
// //////////////////////////////////////
// STL
#include <cassert>
#include <limits>
#include <sstream>
#include <fstream>
#include <string>
// Boost Unit Test Framework (UTF)
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MAIN
#define BOOST_TEST_MODULE OptimiseTestSuite
#include <boost/test/unit_test.hpp>
// StdAir
#include <stdair/basic/BasLogParams.hpp>
#include <stdair/basic/BasDBParams.hpp>
#include <stdair/service/Logger.hpp>
// RMOL
#include <rmol/RMOL_Service.hpp>
#include <rmol/config/rmol-paths.hpp>

namespace boost_utf = boost::unit_test;

// (Boost) Unit Test XML Report
std::ofstream utfReportStream ("bomsforforecaster_utfresults.xml");
```

```

struct UnitTestConfig {
    UnitTestConfig() {
        boost_utf::unit_test_log.set_stream (utfReportStream);
        boost_utf::unit_test_log.set_format (boost_utf::XML);
        boost_utf::unit_test_log.set_threshold_level (boost_utf::log_test_units);
        //boost_utf::unit_test_log.set_threshold_level
        (boost_utf::log_successful_tests);
    }

    ~UnitTestConfig() {
    }
};

namespace RMOL {

    struct BookingClassData {

        // Attributes
        double _bookingCount;
        double _fare;
        double _sellupFactor;
        bool _censorshipFlag;

        // Constructor
        BookingClassData (const double iBookingCount, const double iFare,
                        const double iSellupFactor, const bool iCensorshipFlag)
            : _bookingCount(iBookingCount), _fare(iFare),
              _sellupFactor(iSellupFactor), _censorshipFlag(iCensorshipFlag) {
        }

        // Getters
        double getFare () const {
            return _fare;
        }

        bool getCensorshipFlag () const {
            return _censorshipFlag;
        }

        // Display
        std::string toString() const {
            std::ostringstream ostr;
            ostr << std::endl
                << "[Booking class data information]" << std::endl
                << "Booking counter: " << _bookingCount << std::endl
                << "Fare: " << _fare << std::endl
                << "Sell-up Factor: " << _sellupFactor << std::endl
                << "censorshipFlag: " << _censorshipFlag << std::endl;
            return ostr.str();
        }
    };

    struct BookingClassDataSet {

        typedef std::vector<BookingClassData*> BookingClassDataList_T;

        // Attributes
        int _numberOfClass;
        double _minimumFare;
        bool _censorshipFlag; // true if any of the classes is censored
        BookingClassDataList_T _bookingClassDataList;

        // Constructor
        BookingClassDataSet ()
            : _numberOfClass(0), _minimumFare(0),
              _censorshipFlag(false) {
        }

        // Add BookingClassData
        void addBookingClassData (BookingClassData& ioBookingClassData) {
            _bookingClassDataList.push_back (&ioBookingClassData);
        }

        // Getters
        unsigned int getNumberOfClass () const {
            return _bookingClassDataList.size();
        }

        double getMinimumFare () const {
            return _minimumFare;
        }

        bool getCensorshipFlag () const {
            return _censorshipFlag;
        }
    };
};

```

```

// Setters
void setMinimumFare (const double iMinFare) {
    _minimumFare = iMinFare;
}

void setCensorshipFlag (const bool iCensorshipFlag) {
    _censorshipFlag = iCensorshipFlag;
}

// compute minimum fare
void updateMinimumFare() {
    double minFare = std::numeric_limits<double>::max();
    BookingClassDataList_T::iterator itBookingClassDataList;
    for (itBookingClassDataList = _bookingClassDataList.begin();
        itBookingClassDataList != _bookingClassDataList.end();
        ++itBookingClassDataList) {
        BookingClassData* lBookingClassData = *itBookingClassDataList;
        assert (lBookingClassData != NULL);

        const double lFare = lBookingClassData->getFare();
        if (lFare < minFare) {
            minFare = lFare;
        }
    }
    //
    setMinimumFare(minFare);
}

// compute censorship flag for the data set
void updateCensorshipFlag () {
    bool censorshipFlag = false;
    BookingClassDataList_T::iterator itBookingClassDataList;
    for (itBookingClassDataList = _bookingClassDataList.begin();
        itBookingClassDataList != _bookingClassDataList.end();
        ++itBookingClassDataList) {
        BookingClassData* lBookingClassData = *itBookingClassDataList;
        assert (lBookingClassData != NULL);

        const bool lCensorshipFlagOfAClass =
            lBookingClassData->getCensorshipFlag();
        if (lCensorshipFlagOfAClass) {
            censorshipFlag = true;
            break;
        }
    }
    //
    setCensorshipFlag(censorshipFlag);
}

// Display
std::string toString() const {
    std::ostringstream oStr;
    oStr << std::endl
        << "[Booking class data set information]" << std::endl
        << "Number of classes: " << _numberOfClass << std::endl
        << "Minimum fare: " << _minimumFare << std::endl
        << "The data of the class set are sensed: " << _censorshipFlag
        << std::endl;
    return oStr.str();
}

};

// /**----- BOM : Q-Forecaster ----- */
// struct QForecaster {

//     // Function focused BOM

//     // 1. calculate sell up probability for Q-eq

//     // 2. calculate Q-Equivalent Booking
//     double calculateQEqBooking (BookingClassDataSet& iBookingClassDataSet) {
//         double lQEqBooking = 0.0;
//         double lMinFare = iBookingClassDataSet.getMinimumFare();

//         return lQEqBooking;
//     }

//     /* Calculate Q-equivalent demand
//     [<- performed by unconstrainer if necessary (Using ExpMax BOM)]
//     */

//     // 3. Partition to each class

//     //

```

```

    // };
}

// //////////// Main: Unit Test Suite ////////////

// Set the UTF configuration (re-direct the output to a specific file)
BOOST_GLOBAL_FIXTURE (UnitTestFixture);

BOOST_AUTO_TEST_SUITE (master_test_suite)

BOOST_AUTO_TEST_CASE (rmol_forecaster) {

    // Output log File
    std::string lLogFilename ("bomsforforecaster.log");
    std::ofstream logOutputFile;

    // Open and clean the log outputfile
    logOutputFile.open (lLogFilename.c_str());
    logOutputFile.clear();

    // Initialise the RMOL service
    const stdair::BasLogParams lLogParams (stdair::LOG::DEBUG, logOutputFile);

    // Initialise the RMOL service
    RMOL::RMOL_Service rmolService (lLogParams);

    // Build a sample BOM tree
    rmolService.buildSampleBom();

    // Register BCDataSet
    RMOL::BookingClassDataSet lBookingClassDataSet;

    // Register BookingClassData
    RMOL::BookingClassData QClassData (10, 100, 1, false);
    RMOL::BookingClassData MClassData (5, 150, 0.8, true);
    RMOL::BookingClassData BClassData (0, 200, 0.6, false);
    RMOL::BookingClassData YClassData (0, 300, 0.3, false);

    // Display
    STDAIR_LOG_DEBUG (QClassData.toString());
    STDAIR_LOG_DEBUG (MClassData.toString());
    STDAIR_LOG_DEBUG (BClassData.toString());
    STDAIR_LOG_DEBUG (YClassData.toString());

    // Add BookingClassData into the BCDataSet
    lBookingClassDataSet.addBookingClassData (QClassData);
    lBookingClassDataSet.addBookingClassData (MClassData);
    lBookingClassDataSet.addBookingClassData (BClassData);
    lBookingClassDataSet.addBookingClassData (YClassData);

    // DEBUG
    STDAIR_LOG_DEBUG (lBookingClassDataSet.toString());

    // Number of classes
    const unsigned int lNoOfClass = lBookingClassDataSet.getNumberOfClass();

    // DEBUG
    STDAIR_LOG_DEBUG ("Number of Classes: " << lNoOfClass);

    // Minimum fare
    BOOST_CHECK_NO_THROW (lBookingClassDataSet.updateMinimumFare());
    const double lMinFare = lBookingClassDataSet.getMinimumFare();

    // DEBUG
    STDAIR_LOG_DEBUG ("Minimum fare: " << lMinFare);

    // Censorship flag
    BOOST_CHECK_NO_THROW (lBookingClassDataSet.updateCensorshipFlag());
    const bool lCensorshipFlag = lBookingClassDataSet.getCensorshipFlag();

    // DEBUG
    STDAIR_LOG_DEBUG ("Censorship Flag: " << lCensorshipFlag);

    // Close the log output file
    logOutputFile.close();
}

// End the test suite
BOOST_AUTO_TEST_SUITE_END()

/*!

```

17 Command-Line Test to Demonstrate How To Test the RMOL Project

```

*/
// //////////////////////////////////////
// Import section
// //////////////////////////////////////
// STL
#include <sstream>
#include <fstream>
#include <string>
#include <vector>
#include <cmath>
// Boost Unit Test Framework (UTF)
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MAIN
#define BOOST_TEST_MODULE ForecasterTestSuite
#include <boost/test/unit_test.hpp>
// StdAir
#include <stdair/basic/BasLogParams.hpp>
#include <stdair/basic/BasDBParams.hpp>
#include <stdair/basic/BasFileMgr.hpp>
#include <stdair/service/Logger.hpp>
// RMOL
#include <rmol/RMOL_Service.hpp>

namespace boost_utf = boost::unit_test;

// (Boost) Unit Test XML Report
std::ofstream utfReportStream ("ForecasterTestSuite_utfresults.xml");

struct UnitTestConfig {
    UnitTestConfig() {
        boost_utf::unit_test_log.set_stream (utfReportStream);
        boost_utf::unit_test_log.set_format (boost_utf::XML);
        boost_utf::unit_test_log.set_threshold_level (boost_utf::log_test_units);
        //boost_utf::unit_test_log.set_threshold_level
        (boost_utf::log_successful_tests);
    }

    ~UnitTestConfig() {
    }
};

// ////////////////////////////////// Main: Unit Test Suite //////////////////////////////////

// Set the UTF configuration (re-direct the output to a specific file)
BOOST_GLOBAL_FIXTURE (UnitTestConfig);

BOOST_AUTO_TEST_SUITE (master_test_suite)

BOOST_AUTO_TEST_CASE (rmol_forecaster_q_forecasting) {
    const bool lTestFlag = true; //testForecasterHelper(0);
    BOOST_CHECK_EQUAL (lTestFlag, true);
    BOOST_CHECK_MESSAGE (lTestFlag == true,
        "The test has failed. Please see the log file for "
        "<< \"more details\"");
}

// End the test suite
BOOST_AUTO_TEST_SUITE_END()

/*!

```

18 Command-Line Test to Demonstrate How To Test the RMOL Project

```

*/
// //////////////////////////////////////
// Import section
// //////////////////////////////////////
// STL
#include <sstream>
#include <fstream>
#include <string>
// Boost Unit Test Framework (UTF)
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MAIN
#define BOOST_TEST_MODULE OptimiseTestSuite
#include <boost/test/unit_test.hpp>
// StdAir
#include <stdair/basic/BasLogParams.hpp>

```

```

#include <stdair/basic/BasDBParams.hpp>
#include <stdair/basic/BasFileMgr.hpp>
#include <stdair/service/Logger.hpp>
// RMOL
#include <rmol/RMOL_Service.hpp>
#include <rmol/config/rmol-paths.hpp>

namespace boost_utf = boost::unit_test;

// (Boost) Unit Test XML Report
std::ofstream utfReportStream ("OptimiseTestSuite_utfresults.xml");

struct UnitTestConfig {
    UnitTestConfig() {
        boost_utf::unit_test_log.set_stream (utfReportStream);
        boost_utf::unit_test_log.set_format (boost_utf::XML);
        boost_utf::unit_test_log.set_threshold_level (boost_utf::log_test_units);
        //boost_utf::unit_test_log.set_threshold_level
        (boost_utf::log_successful_tests);
    }

    ~UnitTestConfig() {
    }
};

// ////////////////////////////////////////
int testOptimiseHelper (const unsigned short optimisationMethodFlag,
                       const bool isBuiltin) {

    // Return value
    int oExpectedBookingLimit = 0;

    // Output log File
    std::ostream oStr;
    oStr << "OptimiseTestSuite_" << optimisationMethodFlag << ".log";
    const stdair::Filename_T lLogFilename (oStr.str());

    // Number of random draws to be generated (best if greater than 100)
    const int K = 100000;

    // Methods of optimisation (0 = Monte-Carlo, 1 = Dynamic Programming,
    // 2 = EMSR, 3 = EMSR-a, 4 = EMSR-b, 5 = EMSR-a with sellup prob.)
    const unsigned short METHOD_FLAG = optimisationMethodFlag;

    // Cabin Capacity (it must be greater then 100 here)
    const double cabinCapacity = 100.0;

    // Set the log parameters
    std::ofstream logOutputFile;
    // Open and clean the log outputfile
    logOutputFile.open (lLogFilename.c_str());
    logOutputFile.clear();

    // Initialise the RMOL service
    const stdair::BasLogParams lLogParams (stdair::LOG::DEBUG, logOutputFile);
    RMOL::RMOL_Service rmolService (lLogParams);

    // Check whether or not a (CSV) input file should be read
    if (isBuiltin == true) {

        // Build the default sample BOM tree and build a dummy BOM tree.
        rmolService.buildSampleBom();

    } else {

        // Parse the optimisation data and build a dummy BOM tree
        const stdair::Filename_T lRMInputFileName (STDAIR_SAMPLE_DIR
            "/rm02.csv");
        rmolService.parseAndLoad (cabinCapacity, lRMInputFileName);
    }

    switch (METHOD_FLAG) {
    case 0: {
        // DEBUG
        STDAIR_LOG_DEBUG ("Optimisation by Monte-Carlo (MC)");

        // Calculate the optimal protections by the Monte Carlo
        // Integration approach
        rmolService.optimalOptimisationByMCIntegration (K);
        break;
    }

    case 1: {
        // DEBUG
        STDAIR_LOG_DEBUG ("Optimisation by Dynamic Programming (DP)");
    }
    }
}

```

```

    // Calculate the optimal protections by DP.
    rmolService.optimalOptimisationByDP ();
    break;
}

case 2: {
    // DEBUG
    STDAIR_LOG_DEBUG ("Calculate the Bid-Price Vectors (BPV) by EMSR");

    // Calculate the Bid-Price Vector by EMSR
    rmolService.heuristicOptimisationByEmsr ();
    break;
}

case 3: {
    // DEBUG
    STDAIR_LOG_DEBUG ("Calculate the Authorisation Levels (AUs) by EMSRa");

    // Calculate the protections by EMSR-a
    // Test the EMSR-a algorithm implementation
    rmolService.heuristicOptimisationByEmsrA ();

    // Return a cumulated booking limit value to test
    // oExpectedBookingLimit = static_cast<int> (lBookingLimitVector.at(2));
    break;
}

case 4: {
    // DEBUG
    STDAIR_LOG_DEBUG ("Calculate the Authorisation Levels (AUs) by EMSRb");

    // Calculate the protections by EMSR-b
    rmolService.heuristicOptimisationByEmsrB ();
    break;
}

default: rmolService.optimalOptimisationByMCIntegration (K);
}

// Close the log file
logOutputFile.close();

return oExpectedBookingLimit;
}

// ////////////////////////////////// Main: Unit Test Suite //////////////////////////////////

// Set the UTF configuration (re-direct the output to a specific file)
BOOST_GLOBAL_FIXTURE (UnitTestFixture);

// ////////////////////////////////////////
// Tests are based on the following input values
// price; mean; standard deviation;
// 1050; 17.3; 5.8;
// 567; 45.1; 15.0;
// 534; 39.6; 13.2;
// 520; 34.0; 11.3;
// ////////////////////////////////////////

BOOST_AUTO_TEST_SUITE (master_test_suite)

BOOST_AUTO_TEST_CASE (rmol_optimisation_monte_carlo) {

    // State whether the BOM tree should be built-in or parsed from an input file
    const bool isBuiltin = false;

    BOOST_CHECK_NO_THROW (testOptimiseHelper(0, isBuiltin));
}

BOOST_AUTO_TEST_CASE (rmol_optimisation_dynamic_programming) {

    // State whether the BOM tree should be built-in or parsed from an input file
    const bool isBuiltin = false;

    BOOST_CHECK_NO_THROW (testOptimiseHelper(1, isBuiltin));
}

BOOST_AUTO_TEST_CASE (rmol_optimisation_emsr_bpv) {

    // State whether the BOM tree should be built-in or parsed from an input file
    const bool isBuiltin = false;

    BOOST_CHECK_NO_THROW (testOptimiseHelper(2, isBuiltin));
}

```

```

BOOST_AUTO_TEST_CASE (rmol_optimisation_emsr_a) {

    // State whether the BOM tree should be built-in or parsed from an input file
    const bool isBuiltin = false;

    BOOST_CHECK_NO_THROW (testOptimiseHelper(3, isBuiltin));
    // const int lBookingLimit = testOptimiseHelper(3);
    // const int lExpectedBookingLimit = 61;
    // BOOST_CHECK_EQUAL (lBookingLimit, lExpectedBookingLimit);
    // BOOST_CHECK_MESSAGE (lBookingLimit == lExpectedBookingLimit,
    //                       "The booking limit is " << lBookingLimit
    //                       << ", but it is expected to be "
    //                       << lExpectedBookingLimit);
}

BOOST_AUTO_TEST_CASE (rmol_optimisation_emsr_b) {

    // State whether the BOM tree should be built-in or parsed from an input file
    const bool isBuiltin = false;

    BOOST_CHECK_NO_THROW (testOptimiseHelper(4, isBuiltin));
}

BOOST_AUTO_TEST_CASE (rmol_optimisation_monte_carlo_built_in) {

    // State whether the BOM tree should be built-in or parsed from an input file
    const bool isBuiltin = true;

    BOOST_CHECK_NO_THROW (testOptimiseHelper(5, isBuiltin));
}

BOOST_AUTO_TEST_CASE (rmol_optimisation_dynamic_programming_built_in) {

    // State whether the BOM tree should be built-in or parsed from an input file
    const bool isBuiltin = true;

    BOOST_CHECK_NO_THROW (testOptimiseHelper(6, isBuiltin));
}

BOOST_AUTO_TEST_CASE (rmol_optimisation_emsr_bpv_built_in) {

    // State whether the BOM tree should be built-in or parsed from an input file
    const bool isBuiltin = true;

    BOOST_CHECK_NO_THROW (testOptimiseHelper(7, isBuiltin));
}

BOOST_AUTO_TEST_CASE (rmol_optimisation_emsr_a_built_in) {

    // State whether the BOM tree should be built-in or parsed from an input file
    const bool isBuiltin = true;

    BOOST_CHECK_NO_THROW (testOptimiseHelper(8, isBuiltin));
}

BOOST_AUTO_TEST_CASE (rmol_optimisation_emsr_b_built_in) {

    // State whether the BOM tree should be built-in or parsed from an input file
    const bool isBuiltin = true;

    BOOST_CHECK_NO_THROW (testOptimiseHelper(9, isBuiltin));
}

// End the test suite
BOOST_AUTO_TEST_SUITE_END()

/*!

```

19 Command-Line Test to Demonstrate How To Test the RMOL Project

```

*/
// //////////////////////////////////////
// Import section
// //////////////////////////////////////
// STL
#include <sstream>
#include <fstream>
#include <string>
// Boost Unit Test Framework (UTF)
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MAIN
#define BOOST_TEST_MODULE UnconstrainerTestSuite
#include <boost/test/unit_test.hpp>

```



```

// StdAir
#include <stdair/basic/BasLogParams.hpp>
#include <stdair/basic/BasDBParams.hpp>
#include <stdair/basic/BasFileMgr.hpp>
#include <stdair/service/Logger.hpp>
// RMOL
#include <rmol/RMOL_Service.hpp>

namespace boost_utf = boost::unit_test;

// (Boost) Unit Test XML Report
std::ofstream utfReportStream ("UnconstrainerTestSuite_utfresults.xml");

struct UnitTestConfig {
    UnitTestConfig() {
        boost_utf::unit_test_log.set_stream (utfReportStream);
        boost_utf::unit_test_log.set_format (boost_utf::XML);
        boost_utf::unit_test_log.set_threshold_level (boost_utf::log_test_units);
        //boost_utf::unit_test_log.set_threshold_level
        (boost_utf::log_successful_tests);
    }

    ~UnitTestConfig() {
    }
};

// //////////// Main: Unit Test Suite ////////////

// Set the UTF configuration (re-direct the output to a specific file)
BOOST_GLOBAL_FIXTURE (UnitTestConfig);

BOOST_AUTO_TEST_SUITE (master_test_suite)

BOOST_AUTO_TEST_CASE (rmol_unconstraining_em) {
    const bool lTestFlag = true; // testUnconstrainerHelper(0);
    BOOST_CHECK_EQUAL (lTestFlag, true);
    BOOST_CHECK_MESSAGE (lTestFlag == true,
        "The test has failed. Please see the log file for "
        "<< "more details");
}

// End the test suite
BOOST_AUTO_TEST_SUITE_END ()

/*!

```

20 Namespace Index

20.1 Namespace List

Here is a list of all namespaces with brief descriptions:

RMOL	52
stdair	55

21 Class Index

21.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

```

std::basic_fstream< char >
std::basic_fstream< wchar_t >
std::basic_ifstream< char >
std::basic_ifstream< wchar_t >
std::basic_ios< char >
std::basic_ios< wchar_t >
std::basic_iostream< char >

```

```

std::basic_istream< wchar_t >
std::basic_istream< char >
std::basic_istream< wchar_t >
std::basic_istream< char >
std::basic_istream< wchar_t >
std::basic_ofstream< char >
std::basic_ofstream< wchar_t >
std::basic_ostream< char >
std::basic_ostream< wchar_t >
std::basic_ostringstream< char >
std::basic_ostringstream< wchar_t >
std::basic_string< char >
std::basic_string< wchar_t >
std::basic_stringstream< char >
std::basic_stringstream< wchar_t >

```

CmdAbstract	56
RMOL::InventoryParser	73
RMOL::DefaultDCPList	56
RMOL::DefaultMap	56
RMOL::DemandGeneratorList	57
RMOL::Detruncator	58
RMOL::DPOptimiser	59
RMOL::EMDetruncator	60
RMOL::Emsr	60
RMOL::EmsrUtils	61
FacServiceAbstract	63
RMOL::FacRmolServiceContext	62
RMOL::Forecaster	64
RMOL::GuillotineBlockHelper	66
RMOL::MCOptimiser	73
RMOL::Optimiser	75
RMOL::RMOL_Service	78
RootException	85
RMOL::ForecastException	65
RMOL::OptimisationException	74
RMOL::OverbookingException	78
RMOL::UnconstrainingException	87
ServiceAbstract	85

RMOL::RMOL_ServiceContext	84
StructAbstract	86
RMOL::HistoricalBooking	67
RMOL::HistoricalBookingHolder	69
TestFixture	86
ForecasterTestSuite	64
OptimiseTestSuite	76
UnconstrainerTestSuite	86
RMOL::Utilities	88

22 Class Index

22.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

CmdAbstract	56
RMOL::DefaultDCPList	56
RMOL::DefaultMap	56
RMOL::DemandGeneratorList	57
RMOL::Detruncator	58
RMOL::DPOptimiser	59
RMOL::EMDetruncator	60
RMOL::Emsr	60
RMOL::EmsrUtils	61
RMOL::FacRmolServiceContext	
Factory for the service context	62
FacServiceAbstract	63
RMOL::Forecaster	64
ForecasterTestSuite	64
RMOL::ForecastException	
Forecast-related exception	65
RMOL::GuillotineBlockHelper	66
RMOL::HistoricalBooking	
Structure keeping track, for a given class, of the number of historical bookings and of the censorship flag	67

RMOL::HistoricalBookingHolder	69
RMOL::InventoryParser	
Class filling the virtual class list (representing a list of classes/buckets) from a given input inventory	73
RMOL::MCOptimiser	73
RMOL::OptimisationException	
Optimisation-related exception	74
RMOL::Optimiser	75
OptimiseTestSuite	76
RMOL::OverbookingException	
Overbooking-related exception	78
RMOL::RMOL_Service	
Interface for the RMOL Services	78
RMOL::RMOL_ServiceContext	
Inner class holding the context for the RMOL Service object	84
RootException	85
ServiceAbstract	85
StructAbstract	86
TestFixture	86
UnconstrainerTestSuite	86
RMOL::UnconstrainingException	
Unconstraining-related exception	87
RMOL::Utilities	88

23 File Index

23.1 File List

Here is a list of all files with brief descriptions:

rmol/RMOL_Service.hpp	162
rmol/RMOL_Types.hpp	165
rmol/basic/BasConst.cpp	90
rmol/basic/BasConst_Curves.hpp	91
rmol/basic/BasConst_General.hpp	92
rmol/basic/BasConst_RMOL_Service.hpp	92
rmol/batches/rmol.cpp	95
rmol/bom/BucketHolderTypes.hpp	98

rmol/bom/ DistributionParameterList.hpp	99
rmol/bom/ DPOptimiser.cpp	99
rmol/bom/ DPOptimiser.hpp	102
rmol/bom/ EMDetruncator.cpp	103
rmol/bom/ EMDetruncator.hpp	104
rmol/bom/ Emsr.cpp	105
rmol/bom/ Emsr.hpp	107
rmol/bom/ EmsrUtils.cpp	108
rmol/bom/ EmsrUtils.hpp	109
rmol/bom/ GuillotineBlockHelper.cpp	110
rmol/bom/ GuillotineBlockHelper.hpp	111
rmol/bom/ HistoricalBooking.cpp	112
rmol/bom/ HistoricalBooking.hpp	113
rmol/bom/ HistoricalBookingHolder.cpp	114
rmol/bom/ HistoricalBookingHolder.hpp	118
rmol/bom/ MCOptimiser.cpp	119
rmol/bom/ MCOptimiser.hpp	123
rmol/bom/ Utilities.cpp	126
rmol/bom/ Utilities.hpp	128
rmol/bom/old/ DemandGeneratorList.cpp	124
rmol/bom/old/ DemandGeneratorList.hpp	125
rmol/command/ Detruncator.cpp	129
rmol/command/ Detruncator.hpp	137
rmol/command/ Forecaster.cpp	138
rmol/command/ Forecaster.hpp	149
rmol/command/ InventoryParser.cpp	151
rmol/command/ InventoryParser.hpp	153
rmol/command/ Optimiser.cpp	154
rmol/command/ Optimiser.hpp	157
rmol/config/ rmol-paths.hpp	159
rmol/factory/ FacRmolServiceContext.cpp	160
rmol/factory/ FacRmolServiceContext.hpp	161

rmol/service/RMOL_Service.cpp	167
rmol/service/RMOL_ServiceContext.cpp	191
rmol/service/RMOL_ServiceContext.hpp	192
test/rmol/bomsforforecaster.cpp	193
test/rmol/ForecasterTestSuite.cpp	196
test/rmol/ForecasterTestSuite.hpp	197
test/rmol/OptimiseTestSuite.cpp	198
test/rmol/OptimiseTestSuite.hpp	201
test/rmol/UnconstrainerTestSuite.cpp	202
test/rmol/UnconstrainerTestSuite.hpp	203

24 Namespace Documentation

24.1 RMOL Namespace Reference

Classes

- struct [DefaultMap](#)
- struct [DefaultDCPList](#)
- class [DPOptimiser](#)
- class [EMDetruncator](#)
- class [Emsr](#)
- class [EmsrUtils](#)
- class [GuillotineBlockHelper](#)
- struct [HistoricalBooking](#)

Structure keeping track, for a given class, of the number of historical bookings and of the censorship flag.

- struct [HistoricalBookingHolder](#)
- class [MCOptimiser](#)
- class [DemandGeneratorList](#)
- class [Utilities](#)
- class [Detruncator](#)
- class [Forecaster](#)
- class [InventoryParser](#)

Class filling the virtual class list (representing a list of classes/buckets) from a given input inventory.

- class [Optimiser](#)
- class [FacRmolServiceContext](#)

Factory for the service context.

- class [RMOL_Service](#)

Interface for the [RMOL](#) Services.

- class [OverbookingException](#)

Overbooking-related exception.

- class [UnconstrainingException](#)

Unconstraining-related exception.

- class [ForecastException](#)

Forecast-related exception.

- class [OptimisationException](#)

Optimisation-related exception.

- class [RMOL_ServiceContext](#)

Inner class holding the context for the [RMOL](#) Service object.

Typedefs

- typedef std::list< BucketHolder * > [BucketHolderList_T](#)
- typedef std::list< FldDistributionParameters > [DistributionParameterList_T](#)
- typedef std::vector< [HistoricalBooking](#) > [HistoricalBookingVector_T](#)
- typedef boost::shared_ptr< [RMOL_Service](#) > [RMOL_ServicePtr_T](#)
- typedef std::vector< stdair::NbOfRequests_T > [UnconstrainedDemandVector_T](#)
- typedef std::vector< stdair::NbOfBookings_T > [BookingVector_T](#)
- typedef std::vector< stdair::Flag_T > [FlagVector_T](#)
- typedef std::map< stdair::BookingClass *, [UnconstrainedDemandVector_T](#) > [BookingClassUnconstrainedDemandVectorMap_T](#)
- typedef std::map< stdair::BookingClass *, stdair::NbOfRequests_T > [BookingClassUnconstrainedDemandMap_T](#)
- typedef std::map< const stdair::DTD_T, double > [FRAT5Curve_T](#)

Variables

- const stdair::AirlineCode_T [DEFAULT_RMOL_SERVICE_AIRLINE_CODE](#) = "BA"
- const double [DEFAULT_RMOL_SERVICE_CAPACITY](#) = 1.0
- const int [DEFAULT_NUMBER_OF_DRAWS_FOR_MC_SIMULATION](#) = 100000
- const int [DEFAULT_PRECISION](#) = 10
- const double [DEFAULT_EPSILON](#) = 0.0001
- const double [DEFAULT_STOPPING_CRITERION](#) = 0.01
- const double [DEFAULT_INITIALIZER_DOUBLE_NEGATIVE](#) = -10.0
- const [FRAT5Curve_T](#) [DEFAULT_CUMULATIVE_FRAT5_CURVE](#)
- const stdair::DCPList_T [DEFAULT_DCP_LIST](#) = [DefaultDCPList::init\(\)](#)

24.1.1 Typedef Documentation

24.1.1.1 typedef std::list<BucketHolder*> RMOL::BucketHolderList_T

Define a vector (ordered list) of N bucket/classe holders.

Definition at line 16 of file [BucketHolderTypes.hpp](#).

24.1.1.2 typedef std::list<FldDistributionParameters> RMOL::DistributionParameterList_T

Define the set of parameters, each of one wrapping a pair of distribution parameters (i.e., mean and standard deviation).

Definition at line 16 of file [DistributionParameterList.hpp](#).

24.1.1.3 typedef std::vector<HistoricalBooking> RMOL::HistoricalBookingVector_T

Define a vector (ordered list) of N HistoricalBookings.

Definition at line 16 of file [HistoricalBookingHolder.hpp](#).

24.1.1.4 typedef boost::shared_ptr<RMOL_Service> RMOL::RMOL_ServicePtr_T

Pointer on the [RMOL](#) Service handler.

Definition at line 73 of file [RMOL_Types.hpp](#).

24.1.1.5 typedef std::vector<stdair::NbOfRequests_T> RMOL::UnconstrainedDemandVector_T

Define the vector of historical unconstrained demand.

Definition at line 76 of file [RMOL_Types.hpp](#).

24.1.1.6 typedef std::vector<stdair::NbOfBookings_T> RMOL::BookingVector_T

Define the vector of historical bookings.

Definition at line 79 of file [RMOL_Types.hpp](#).

24.1.1.7 typedef std::vector<stdair::Flag_T> RMOL::FlagVector_T

Define the vector of censorship flags.

Definition at line 82 of file [RMOL_Types.hpp](#).

**24.1.1.8 typedef std::map<stdair::BookingClass*, UnconstrainedDemandVector_T>
RMOL::BookingClassUnconstrainedDemandVectorMap_T**

Define the map between the booking class and it's corresponding unconstrained demand vector.

Definition at line 86 of file [RMOL_Types.hpp](#).

**24.1.1.9 typedef std::map<stdair::BookingClass*, stdair::NbOfRequests_T> RMOL::BookingClassUnconstrained-
DemandMap_T**

Define the map between the booking class and it's corresponding unconstrained demand.

Definition at line 90 of file [RMOL_Types.hpp](#).

24.1.1.10 typedef std::map<const stdair::DTD_T, double> RMOL::FRAT5Curve_T

Define the FRAT5 curve.

Definition at line 93 of file [RMOL_Types.hpp](#).

24.1.2 Variable Documentation**24.1.2.1 const stdair::AirlineCode_T RMOL::DEFAULT_RMOL_SERVICE_AIRLINE_CODE = "BA"**

Default airline code for the [RMOL_Service](#).

Definition at line 11 of file [BasConst.cpp](#).

24.1.2.2 const double RMOL::DEFAULT_RMOL_SERVICE_CAPACITY = 1.0

Default capacity for the [RMOL_Service](#).

Definition at line 14 of file [BasConst.cpp](#).

24.1.2.3 `const int RMOL::DEFAULT_NUMBER_OF_DRAWS_FOR_MC_SIMULATION = 100000`

Default value for the number of draws within the Monte-Carlo Integration algorithm.

Definition at line 18 of file [BasConst.cpp](#).

24.1.2.4 `const int RMOL::DEFAULT_PRECISION = 10`

Default value for the precision of the integral computation in the Dynamic Programming algorithm (100 means that the precision will be 0.01).

Default value for the precision of the integral computation in the Dynamic Programming algorithm.

Definition at line 23 of file [BasConst.cpp](#).

24.1.2.5 `const double RMOL::DEFAULT_EPSILON = 0.0001`

Default epsilon value to qualify a denominator

Definition at line 26 of file [BasConst.cpp](#).

24.1.2.6 `const double RMOL::DEFAULT_STOPPING_CRITERION = 0.01`

Default stopping value for an iterative algorithm.

Definition at line 29 of file [BasConst.cpp](#).

24.1.2.7 `const double RMOL::DEFAULT_INITIALIZER_DOUBLE_NEGATIVE = -10.0`

Default negative value used to initialize a double variable.

Definition at line 32 of file [BasConst.cpp](#).

24.1.2.8 `const FRAT5Curve_T RMOL::DEFAULT_CUMULATIVE_FRAT5_CURVE`

Initial value:

```
DefaultMap::createCumulativeFRAT5Curve()
```

Default cumulative[for the remaining period] FRAT5 curve for forecasting and optimisation.

Default cumulative (for the remaining period) FRAT5 curve for forecasting and optimisation.

Definition at line 36 of file [BasConst.cpp](#).

24.1.2.9 `const stdair::DCPList_T RMOL::DEFAULT_DCP_LIST = DefaultDCPList::init()`

Default data collection point list.

Definition at line 69 of file [BasConst.cpp](#).

Referenced by [RMOL::Utilities::buildRemainingDCPList\(\)](#), [RMOL::Utilities::buildRemainingDCPList2\(\)](#), [RMOL::RMOL_Service::forecastOnD\(\)](#), [RMOL::RMOL_Service::optimiseOnD\(\)](#), [RMOL::RMOL_Service::optimiseOnDUsingAdvancedRMCooperation\(\)](#), [RMOL::RMOL_Service::optimiseOnDUsingRMCooperation\(\)](#), [RMOL::RMOL_Service::projectAggregatedDemandOnLegCabins\(\)](#), [RMOL::RMOL_Service::projectOnDDemandOnLegCabinsUsingDA\(\)](#), [RMOL::RMOL_Service::projectOnDDemandOnLegCabinsUsingDYP\(\)](#), [RMOL::RMOL_Service::projectOnDDemandOnLegCabinsUsingYP\(\)](#), [RMOL::RMOL_Service::resetDemandInformation\(\)](#), and [RMOL::RMOL_Service::updateBidPrice\(\)](#).

24.2 stdair Namespace Reference

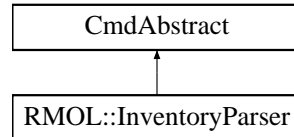
24.2.1 Detailed Description

Forward declarations.

25 Class Documentation

25.1 CmdAbstract Class Reference

Inheritance diagram for CmdAbstract:



The documentation for this class was generated from the following file:

- [rmol/command/InventoryParser.hpp](#)

25.2 RMOL::DefaultDCPList Struct Reference

```
#include <rmol/basic/BasConst_General.hpp>
```

Static Public Member Functions

- static `stdair::DCPList_T` [init](#) ()

25.2.1 Detailed Description

Definition at line 31 of file [BasConst_General.hpp](#).

25.2.2 Member Function Documentation

25.2.2.1 `stdair::DCPList_T` [RMOL::DefaultDCPList::init](#) () [static]

Definition at line 70 of file [BasConst.cpp](#).

The documentation for this struct was generated from the following files:

- [rmol/basic/BasConst_General.hpp](#)
- [rmol/basic/BasConst.cpp](#)

25.3 RMOL::DefaultMap Struct Reference

```
#include <rmol/basic/BasConst_Curves.hpp>
```

Static Public Member Functions

- static `FRAT5Curve_T` [createCumulativeFRAT5Curve](#) ()

25.3.1 Detailed Description

Default PoS probability mass.

Definition at line 17 of file [BasConst_Curves.hpp](#).

25.3.2 Member Function Documentation

25.3.2.1 FRAT5Curve_T RMOL::DefaultMap::createCumulativeFRAT5Curve () [static]

Definition at line 38 of file [BasConst.cpp](#).

The documentation for this struct was generated from the following files:

- [rmol/basic/BasConst_Curves.hpp](#)
- [rmol/basic/BasConst.cpp](#)

25.4 RMOL::DemandGeneratorList Class Reference

```
#include <rmol/bom/old/DemandGeneratorList.hpp>
```

Public Member Functions

- [DemandGeneratorList](#) ()
- [DemandGeneratorList](#) (const [DemandGeneratorList](#) &)
- [DemandGeneratorList](#) (const [DistributionParameterList_T](#) &)
- virtual [~DemandGeneratorList](#) ()
- void [generateVariateList](#) ([VariateList_T](#) &) const

Protected Types

- typedef std::list< [Gaussian](#) > [DemandGeneratorList_T](#)

25.4.1 Detailed Description

Wrapper around a set of Gaussian Random Generators.

Definition at line 17 of file [DemandGeneratorList.hpp](#).

25.4.2 Member Typedef Documentation

25.4.2.1 typedef std::list<Gaussian> RMOL::DemandGeneratorList::DemandGeneratorList_T [protected]

Define a (ordered) set of Gaussian Random Generators.

Definition at line 20 of file [DemandGeneratorList.hpp](#).

25.4.3 Constructor & Destructor Documentation

25.4.3.1 RMOL::DemandGeneratorList::DemandGeneratorList ()

Constructors.

Definition at line 10 of file [DemandGeneratorList.cpp](#).

25.4.3.2 RMOL::DemandGeneratorList::DemandGeneratorList (const [DemandGeneratorList](#) & *iDemandGeneratorList*)

Definition at line 17 of file [DemandGeneratorList.cpp](#).

25.4.3.3 RMOL::DemandGeneratorList::DemandGeneratorList (const DistributionParameterList_T & iDistributionParameterList)

List of distribution parameters (mean, standard deviation).

Definition at line 25 of file [DemandGeneratorList.cpp](#).

25.4.3.4 RMOL::DemandGeneratorList::~~DemandGeneratorList () [virtual]

Destructors.

Definition at line 30 of file [DemandGeneratorList.cpp](#).

25.4.4 Member Function Documentation

25.4.4.1 void RMOL::DemandGeneratorList::generateVariateList (VariateList_T & ioVariateList) const

Definition at line 50 of file [DemandGeneratorList.cpp](#).

The documentation for this class was generated from the following files:

- [rmol/bom/old/DemandGeneratorList.hpp](#)
- [rmol/bom/old/DemandGeneratorList.cpp](#)

25.5 RMOL::Detruncator Class Reference

```
#include <rmol/command/Detruncator.hpp>
```

Static Public Member Functions

- static void [unconstrainUsingAdditivePickUp](#) (const stdair::SegmentCabin &, [BookingClassUnconstrainedDemandVectorMap_T](#) &, [UnconstrainedDemandVector_T](#) &, const stdair::DCP_T &, const stdair::DCP_T &, const stdair::Date_T &)
- static void [unconstrainUsingMultiplicativePickUp](#) (const stdair::SegmentCabin &, [BookingClassUnconstrainedDemandVectorMap_T](#) &, [UnconstrainedDemandVector_T](#) &, const stdair::DCP_T &, const stdair::DCP_T &, const stdair::Date_T &, const stdair::NbOfSegments_T &)
- static void [retrieveUnconstrainedDemandForFirstDCP](#) (const stdair::SegmentCabin &, [BookingClassUnconstrainedDemandVectorMap_T](#) &, [UnconstrainedDemandVector_T](#) &, const stdair::DCP_T &, const stdair::NbOfSegments_T &, const stdair::NbOfSegments_T &)
- static void [unconstrainUsingMultiplicativePickUp](#) ([HistoricalBookingHolder](#) &)

25.5.1 Detailed Description

Class wrapping the principal unconstraining algorithms and some accessory algorithms.

Definition at line 24 of file [Detruncator.hpp](#).

25.5.2 Member Function Documentation

25.5.2.1 void RMOL::Detruncator::unconstrainUsingAdditivePickUp (const stdair::SegmentCabin & iSegmentCabin, [BookingClassUnconstrainedDemandVectorMap_T](#) & ioBkgClassUncDemMap, [UnconstrainedDemandVector_T](#) & ioQEquivalentDemandVector, const stdair::DCP_T & iDCPBegin, const stdair::DCP_T & iDCPEnd, const stdair::Date_T & iCurrentDate) [static]

Unconstrain booking figures between two DCP's.

Definition at line 25 of file [Detruncator.cpp](#).

References [RMOL::GuillotineBlockHelper::getNbOfSegmentAlreadyPassedThisDTD\(\)](#).

25.5.2.2 void RMOL::Detruncator::unconstrainUsingMultiplicativePickUp (const stdair::SegmentCabin & *iSegmentCabin*, BookingClassUnconstrainedDemandVectorMap_T & *ioBkgClassUncDemMap*, UnconstrainedDemandVector_T & *ioQEquivalentDemandVector*, const stdair::DCP_T & *iDCPBegin*, const stdair::DCP_T & *iDCPEnd*, const stdair::Date_T & *iCurrentDate*, const stdair::NbOfSegments_T & *iNbOfDepartedSegments*) [static]

Unconstrain booking figures between two DCP's.

Definition at line 317 of file [Detruncator.cpp](#).

References [RMOL::GuillotineBlockHelper::getNbOfSegmentAlreadyPassedThisDTD\(\)](#).

25.5.2.3 void RMOL::Detruncator::retrieveUnconstrainedDemandForFirstDCP (const stdair::SegmentCabin & *iSegmentCabin*, BookingClassUnconstrainedDemandVectorMap_T & *ioBkgClassUncDemVectorMap*, UnconstrainedDemandVector_T & *ioQEquivalentDemandVector*, const stdair::DCP_T & *iFirstDCP*, const stdair::NbOfSegments_T & *iNbOfSegments*, const stdair::NbOfSegments_T & *iNbOfUsedSegments*) [static]

Retrieve unconstrained demand figures for the first DCP.

Definition at line 239 of file [Detruncator.cpp](#).

25.5.2.4 void RMOL::Detruncator::unconstrainUsingMultiplicativePickUp (HistoricalBookingHolder & *ioHBHolder*) [static]

Unconstrain the product-oriented booking figures for a given class ou Q-equivalent class.

Definition at line 558 of file [Detruncator.cpp](#).

References [RMOL::HistoricalBookingHolder::getCensorshipFlag\(\)](#), [RMOL::HistoricalBookingHolder::getHistoricalBooking\(\)](#), [RMOL::HistoricalBookingHolder::getNbOfFlights\(\)](#), [RMOL::HistoricalBookingHolder::getNbOfUncensoredData\(\)](#), and [RMOL::HistoricalBookingHolder::setUnconstrainedDemand\(\)](#).

The documentation for this class was generated from the following files:

- [rmol/command/Detruncator.hpp](#)
- [rmol/command/Detruncator.cpp](#)

25.6 RMOL::DPOptimiser Class Reference

```
#include <rmol/bom/DPOptimiser.hpp>
```

Static Public Member Functions

- static void [optimalOptimisationByDP](#) (stdair::LegCabin &)
- static double [cdfGaussianQ](#) (const double, const double)

25.6.1 Detailed Description

Utility methods for the Dynamic Programming algorithms.

Definition at line 17 of file [DPOptimiser.hpp](#).

25.6.2 Member Function Documentation

25.6.2.1 void RMOL::DPOptimiser::optimalOptimisationByDP (stdair::LegCabin & *ioLegCabin*) [static]

Dynamic Programming to compute the cumulative protection levels and booking limits (described in the book Revenue Management - Talluri & Van Ryzin, p.41-42).

Definition at line 22 of file [DPOptimiser.cpp](#).

25.6.2.2 static double RMOL::DPOptimiser::cdfGaussianQ (const double , const double) [static]

Compute the cdf_Q of a gaussian.

The documentation for this class was generated from the following files:

- [rmol/bom/DPOptimiser.hpp](#)
- [rmol/bom/DPOptimiser.cpp](#)

25.7 RMOL::EMDetruncator Class Reference

```
#include <rmol/bom/EMDetruncator.hpp>
```

Static Public Member Functions

- static void [unconstrainUsingEMMethod](#) (HistoricalBookingHolder &)

25.7.1 Detailed Description

Utility for the Expectation-Maximisation algorithm.

Definition at line 12 of file [EMDetruncator.hpp](#).

25.7.2 Member Function Documentation

25.7.2.1 void RMOL::EMDetruncator::unconstrainUsingEMMethod (HistoricalBookingHolder & *ioHistoricalBookingHolder*) [static]

Unconstrain the censored booking data using the Expection-Maximisation algorithm.

Definition at line 20 of file [EMDetruncator.cpp](#).

References [RMOL::HistoricalBookingHolder::getDemandMean\(\)](#), [RMOL::HistoricalBookingHolder::getListOfToBeUnconstrainedFlags\(\)](#), [RMOL::HistoricalBookingHolder::getNbOffFlights\(\)](#), [RMOL::HistoricalBookingHolder::getNbOfUncensoredBookings\(\)](#), [RMOL::HistoricalBookingHolder::getNbOfUncensoredData\(\)](#), [RMOL::HistoricalBookingHolder::getStandardDeviation\(\)](#), [RMOL::HistoricalBookingHolder::getUncensoredStandardDeviation\(\)](#), [RMOL::HistoricalBookingHolder::getUnconstrainedDemand\(\)](#), and [RMOL::HistoricalBookingHolder::setUnconstrainedDemand\(\)](#).

The documentation for this class was generated from the following files:

- [rmol/bom/EMDetruncator.hpp](#)
- [rmol/bom/EMDetruncator.cpp](#)

25.8 RMOL::Emsr Class Reference

```
#include <rmol/bom/Emsr.hpp>
```

Static Public Member Functions

- static void [heuristicOptimisationByEmsr](#) (stdair::LegCabin &)
- static void [heuristicOptimisationByEmsrA](#) (stdair::LegCabin &)
- static void [heuristicOptimisationByEmsrB](#) (stdair::LegCabin &)

25.8.1 Detailed Description

Class Implementing the EMSR algorithm for Bid-Price Vector computing.

Definition at line 18 of file [Emsr.hpp](#).

25.8.2 Member Function Documentation

25.8.2.1 void RMOL::Emsr::heuristicOptimisationByEmsr (stdair::LegCabin & ioLegCabin) [static]

Compute the Bid-Price Vector using the EMSR algorithm. Then compute the protection levels and booking limits by using the BPV.

For each class/bucket j with yield p_j and demand D_j , compute $p_j \cdot \Pr(D_j \geq x)$ with x the capacity index. This value is called the EMSR (Expected Marginal Seat Revenue) of the class/bucket j with the remaining capacity of x . Thus, we have for each class/bucket a list of EMSR values. We merge all these lists and sort the values from high to low in order to obtain the BPV.

Definition at line 108 of file [Emsr.cpp](#).

References [RMOL::EmsrUtils::computeEmsrValue\(\)](#).

25.8.2.2 void RMOL::Emsr::heuristicOptimisationByEmsrA (stdair::LegCabin & ioLegCabin) [static]

Calculate the optimal protections for the set of buckets/classes given in input, and update those buckets accordingly.

Definition at line 21 of file [Emsr.cpp](#).

References [RMOL::EmsrUtils::computeProtectionLevel\(\)](#).

25.8.2.3 void RMOL::Emsr::heuristicOptimisationByEmsrB (stdair::LegCabin & ioLegCabin) [static]

Compute the protection levels and booking limites by using the EMSR-b algorithm.

Definition at line 64 of file [Emsr.cpp](#).

References [RMOL::EmsrUtils::computeAggregatedVirtualClass\(\)](#), and [RMOL::EmsrUtils::computeProtectionLevel\(\)](#).

The documentation for this class was generated from the following files:

- [rmol/bom/Emsr.hpp](#)
- [rmol/bom/Emsr.cpp](#)

25.9 RMOL::EmsrUtils Class Reference

```
#include <rmol/bom/EmsrUtils.hpp>
```

Static Public Member Functions

- static void [computeAggregatedVirtualClass](#) (stdair::VirtualClassStruct &, stdair::VirtualClassStruct &)
- static const stdair::ProtectionLevel_T [computeProtectionLevel](#) (stdair::VirtualClassStruct &, stdair::VirtualClassStruct &)
- static const double [computeEmsrValue](#) (double, stdair::VirtualClassStruct &)

25.9.1 Detailed Description

Forward declarations.

Definition at line 19 of file [EmsrUtils.hpp](#).

25.9.2 Member Function Documentation

25.9.2.1 `void RMOL::EmsrUtils::computeAggregatedVirtualClass (stdair::VirtualClassStruct & ioAggregatedVirtualClass, stdair::VirtualClassStruct & ioCurrentVirtualClass) [static]`

Compute the aggregated class/bucket of classes/buckets 1,...j for EMSR-b algorithm.

Definition at line 19 of file [EmsrUtils.cpp](#).

Referenced by [RMOL::Emsr::heuristicOptimisationByEmsrB\(\)](#).

25.9.2.2 `const stdair::ProtectionLevel_T RMOL::EmsrUtils::computeProtectionLevel (stdair::VirtualClassStruct & ioAggregatedVirtualClass, stdair::VirtualClassStruct & ioNextVirtualClass) [static]`

Compute the protection level using the Little-Wood formular.

Definition at line 53 of file [EmsrUtils.cpp](#).

Referenced by [RMOL::Emsr::heuristicOptimisationByEmsrA\(\)](#), and [RMOL::Emsr::heuristicOptimisationByEmsrB\(\)](#).

25.9.2.3 `const double RMOL::EmsrUtils::computeEmsrValue (double iCapacity, stdair::VirtualClassStruct & ioVirtualClass) [static]`

Compute the EMSR value of a class/bucket.

Definition at line 80 of file [EmsrUtils.cpp](#).

Referenced by [RMOL::Emsr::heuristicOptimisationByEmsr\(\)](#).

The documentation for this class was generated from the following files:

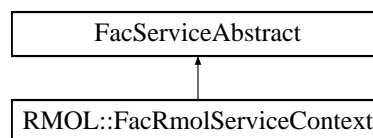
- [rmol/bom/EmsrUtils.hpp](#)
- [rmol/bom/EmsrUtils.cpp](#)

25.10 RMOL::FacRmolServiceContext Class Reference

Factory for the service context.

```
#include <rmol/factory/FacRmolServiceContext.hpp>
```

Inheritance diagram for RMOL::FacRmolServiceContext:



Public Member Functions

- [~FacRmolServiceContext \(\)](#)
- [RMOL_ServiceContext & create \(\)](#)

Static Public Member Functions

- static [FacRmolServiceContext & instance \(\)](#)

Protected Member Functions

- [FacRmolServiceContext \(\)](#)

25.10.1 Detailed Description

Factory for the service context.

Definition at line 22 of file [FacRmolServiceContext.hpp](#).

25.10.2 Constructor & Destructor Documentation

25.10.2.1 RMOL::FacRmolServiceContext::~~FacRmolServiceContext ()

Destructor.

The Destruction put the `_instance` to NULL in order to be clean for the next `FacSimfqtServiceContext::instance()`.

Definition at line 17 of file [FacRmolServiceContext.cpp](#).

25.10.2.2 RMOL::FacRmolServiceContext::FacRmolServiceContext () [inline], [protected]

Default Constructor.

This constructor is protected in order to ensure the singleton pattern.

Definition at line 57 of file [FacRmolServiceContext.hpp](#).

Referenced by [instance\(\)](#).

25.10.3 Member Function Documentation

25.10.3.1 FacRmolServiceContext & RMOL::FacRmolServiceContext::instance () [static]

Provide the unique instance.

The singleton is instantiated when first used.

Returns

FacServiceContext&

Definition at line 22 of file [FacRmolServiceContext.cpp](#).

References [FacRmolServiceContext\(\)](#).

25.10.3.2 RMOL_ServiceContext & RMOL::FacRmolServiceContext::create ()

Create a new ServiceContext object.

This new object is added to the list of instantiated objects.

Returns

ServiceContext& The newly created object.

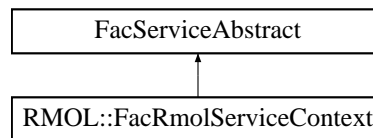
Definition at line 34 of file [FacRmolServiceContext.cpp](#).

The documentation for this class was generated from the following files:

- [rmol/factory/FacRmolServiceContext.hpp](#)
- [rmol/factory/FacRmolServiceContext.cpp](#)

25.11 FacServiceAbstract Class Reference

Inheritance diagram for FacServiceAbstract:



The documentation for this class was generated from the following file:

- [rmol/factory/FacRmolServiceContext.hpp](#)

25.12 RMOL::Forecaster Class Reference

```
#include <rmol/command/Forecaster.hpp>
```

Static Public Member Functions

- static bool [forecastUsingAdditivePickUp](#) (stdair::FlightDate &, const stdair::DateTime_T &)
- static bool [forecastUsingMultiplicativePickUp](#) (stdair::FlightDate &, const stdair::DateTime_T &)

25.12.1 Detailed Description

Class wrapping the optimisation algorithms.

Definition at line 23 of file [Forecaster.hpp](#).

25.12.2 Member Function Documentation

25.12.2.1 bool RMOL::Forecaster::forecastUsingAdditivePickUp (stdair::FlightDate & *ioFlightDate*, const stdair::DateTime_T & *iEventTime*) [static]

Forecast demand for a flight-date using additive pick-up method.

Definition at line 35 of file [Forecaster.cpp](#).

References [RMOL::Utilities::buildRemainingDCPList\(\)](#), and [RMOL::Utilities::buildRemainingDCPList2\(\)](#).

Referenced by [RMOL::RMOL_Service::optimise\(\)](#).

25.12.2.2 bool RMOL::Forecaster::forecastUsingMultiplicativePickUp (stdair::FlightDate & *ioFlightDate*, const stdair::DateTime_T & *iEventTime*) [static]

Forecast demand for a flight-date using multiplicative pick-up method.

Definition at line 276 of file [Forecaster.cpp](#).

Referenced by [RMOL::RMOL_Service::optimise\(\)](#).

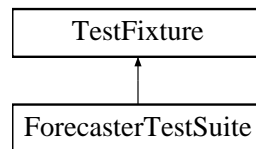
The documentation for this class was generated from the following files:

- [rmol/command/Forecaster.hpp](#)
- [rmol/command/Forecaster.cpp](#)

25.13 ForecasterTestSuite Class Reference

```
#include <test/rmol/ForecasterTestSuite.hpp>
```

Inheritance diagram for ForecasterTestSuite:



Public Member Functions

- void [testQForecaster](#) ()
- [ForecasterTestSuite](#) ()

Protected Attributes

- std::stringstream [_describeKey](#)

25.13.1 Detailed Description

Definition at line 6 of file [ForecasterTestSuite.hpp](#).

25.13.2 Constructor & Destructor Documentation

25.13.2.1 ForecasterTestSuite::ForecasterTestSuite ()

Constructor.

25.13.3 Member Function Documentation

25.13.3.1 void ForecasterTestSuite::testQForecaster ()

Test Q-forecaster.

25.13.4 Member Data Documentation

25.13.4.1 std::stringstream ForecasterTestSuite::_describeKey [protected]

Definition at line 19 of file [ForecasterTestSuite.hpp](#).

The documentation for this class was generated from the following file:

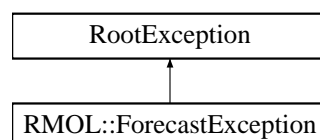
- test/rmol/[ForecasterTestSuite.hpp](#)

25.14 RMOL::ForecastException Class Reference

Forecast-related exception.

```
#include <rmol/RMOL_Types.hpp>
```

Inheritance diagram for RMOL::ForecastException:



Public Member Functions

- [ForecastException](#) (const std::string &iWhat)

25.14.1 Detailed Description

Forecast-related exception.

Definition at line 51 of file [RMOL_Types.hpp](#).

25.14.2 Constructor & Destructor Documentation

25.14.2.1 RMOL::ForecastException::ForecastException (const std::string & iWhat) [inline]

Constructor.

Definition at line 54 of file [RMOL_Types.hpp](#).

The documentation for this class was generated from the following file:

- [rmol/RMOL_Types.hpp](#)

25.15 RMOL::GuillotineBlockHelper Class Reference

```
#include <rmol/bom/GuillotineBlockHelper.hpp>
```

Static Public Member Functions

- static stdair::NbOfSegments_T [getNbOfSegmentAlreadyPassedThisDTD](#) (const stdair::GuillotineBlock &, const stdair::DTD_T &, const stdair::Date_T &)
- static bool [hasPassedThisDTD](#) (const stdair::SegmentCabin &, const stdair::DTD_T &, const stdair::Date_T &)

25.15.1 Detailed Description

Class representing the actual business functions for an airline guillotine block.

Definition at line 23 of file [GuillotineBlockHelper.hpp](#).

25.15.2 Member Function Documentation

25.15.2.1 stdair::NbOfSegments_T RMOL::GuillotineBlockHelper::getNbOfSegmentAlreadyPassedThisDTD (const stdair::GuillotineBlock & iGB, const stdair::DTD_T & iDTD, const stdair::Date_T & iCurrentDate) [static]

Retrieve the number of similar segments which already passed the given DTD.

Definition at line 20 of file [GuillotineBlockHelper.cpp](#).

References [hasPassedThisDTD\(\)](#).

Referenced by [RMOL::Utilities::getNbOfDepartedSimilarSegments\(\)](#), [RMOL::Detruncator::unconstrainUsingAdditivePickUp\(\)](#), and [RMOL::Detruncator::unconstrainUsingMultiplicativePickUp\(\)](#).

25.15.2.2 bool RMOL::GuillotineBlockHelper::hasPassedThisDTD (const stdair::SegmentCabin & iSegmentCabin, const stdair::DTD_T & iDTD, const stdair::Date_T & iCurrentDate) [static]

Check if the given segment has passed the given DTD.

Definition at line 42 of file [GuillotineBlockHelper.cpp](#).

Referenced by [getNbOfSegmentAlreadyPassedThisDTD\(\)](#).

The documentation for this class was generated from the following files:

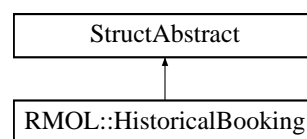
- [rmol/bom/GuillotineBlockHelper.hpp](#)
- [rmol/bom/GuillotineBlockHelper.cpp](#)

25.16 RMOL::HistoricalBooking Struct Reference

Structure keeping track, for a given class, of the number of historical bookings and of the censorship flag.

```
#include <rmol/bom/HistoricalBooking.hpp>
```

Inheritance diagram for RMOL::HistoricalBooking:



Public Member Functions

- `const stdair::NbOfBookings_T & getNbOfBookings () const`
- `const stdair::NbOfBookings_T & getUnconstrainedDemand () const`
- `const stdair::Flag_T & getFlag () const`
- `void setUnconstrainedDemand (const stdair::NbOfBookings_T &iDemand)`
- `void setParameters (const stdair::NbOfBookings_T, const stdair::Flag_T)`
- `void toStream (std::ostream &ioOut) const`
- `const std::string describe () const`
- `void display () const`
- `HistoricalBooking (const stdair::NbOfBookings_T, const stdair::Flag_T)`
- `HistoricalBooking ()`
- `HistoricalBooking (const HistoricalBooking &)`
- `virtual ~HistoricalBooking ()`

25.16.1 Detailed Description

Structure keeping track, for a given class, of the number of historical bookings and of the censorship flag.

Definition at line 17 of file [HistoricalBooking.hpp](#).

25.16.2 Constructor & Destructor Documentation

25.16.2.1 RMOL::HistoricalBooking::HistoricalBooking (const stdair::NbOfBookings_T *iNbOfBookings*, const stdair::Flag_T *iFlag*)

Main constructor.

Definition at line 21 of file [HistoricalBooking.cpp](#).

25.16.2.2 RMOL::HistoricalBooking::HistoricalBooking ()

Default constructor.

Definition at line 15 of file [HistoricalBooking.cpp](#).

25.16.2.3 RMOL::HistoricalBooking::HistoricalBooking (const HistoricalBooking & iHistoricalBooking)

Copy constructor.

Definition at line 29 of file [HistoricalBooking.cpp](#).

25.16.2.4 RMOL::HistoricalBooking::~~HistoricalBooking () [virtual]

Destructor.

Definition at line 36 of file [HistoricalBooking.cpp](#).

25.16.3 Member Function Documentation

25.16.3.1 const stdair::NbOfBookings_T& RMOL::HistoricalBooking::getNbOfBookings () const [inline]

Getter for the booking.

Definition at line 22 of file [HistoricalBooking.hpp](#).

Referenced by [RMOL::HistoricalBookingHolder::calculateExpectedDemand\(\)](#), [RMOL::HistoricalBookingHolder::getHistoricalBooking\(\)](#), [RMOL::HistoricalBookingHolder::getNbOfUncensoredBookings\(\)](#), [RMOL::HistoricalBookingHolder::getUncensoredStandardDeviation\(\)](#), [toStream\(\)](#), and [RMOL::HistoricalBookingHolder::toStream\(\)](#).

25.16.3.2 const stdair::NbOfBookings_T& RMOL::HistoricalBooking::getUnconstrainedDemand () const [inline]

Getter for the unconstrained bookings.

Definition at line 26 of file [HistoricalBooking.hpp](#).

Referenced by [RMOL::HistoricalBookingHolder::getDemandMean\(\)](#), [RMOL::HistoricalBookingHolder::getStandardDeviation\(\)](#), [RMOL::HistoricalBookingHolder::getUnconstrainedDemand\(\)](#), [toStream\(\)](#), and [RMOL::HistoricalBookingHolder::toStream\(\)](#).

25.16.3.3 const stdair::Flag_T& RMOL::HistoricalBooking::getFlag () const [inline]

Getter for the flag of censorship: "false" means that the bookings are not censored.

Definition at line 31 of file [HistoricalBooking.hpp](#).

Referenced by [RMOL::HistoricalBookingHolder::getCensorshipFlag\(\)](#), [RMOL::HistoricalBookingHolder::getListOfToBeUnconstrainedFlags\(\)](#), [RMOL::HistoricalBookingHolder::getNbOfUncensoredBookings\(\)](#), [toStream\(\)](#), and [RMOL::HistoricalBookingHolder::toStream\(\)](#).

25.16.3.4 void RMOL::HistoricalBooking::setUnconstrainedDemand (const stdair::NbOfBookings_T & iDemand) [inline]

Setter for the unconstraining demand.

Definition at line 38 of file [HistoricalBooking.hpp](#).

25.16.3.5 void RMOL::HistoricalBooking::setParameters (const stdair::NbOfBookings_T iNbOfBookings, const stdair::Flag_T iFlag)

Setter for all parameters.

Definition at line 41 of file [HistoricalBooking.cpp](#).

25.16.3.6 void RMOL::HistoricalBooking::toStream (std::ostream & ioOut) const

Dump a Business Object into an output stream.

Parameters

<i>ostream&</i>	the output stream
---------------------	-------------------

Returns

ostream& the output stream.

Definition at line 57 of file [HistoricalBooking.cpp](#).

References [getFlag\(\)](#), [getNbOfBookings\(\)](#), and [getUnconstrainedDemand\(\)](#).

Referenced by [display\(\)](#).

25.16.3.7 const std::string RMOL::HistoricalBooking::describe () const

Give a description of the structure (for display purposes).

Definition at line 48 of file [HistoricalBooking.cpp](#).

25.16.3.8 void RMOL::HistoricalBooking::display () const

Display on standard output.

Definition at line 66 of file [HistoricalBooking.cpp](#).

References [toStream\(\)](#).

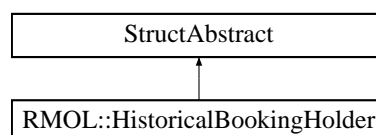
The documentation for this struct was generated from the following files:

- [rmol/bom/HistoricalBooking.hpp](#)
- [rmol/bom/HistoricalBooking.cpp](#)

25.17 RMOL::HistoricalBookingHolder Struct Reference

```
#include <rmol/bom/HistoricalBookingHolder.hpp>
```

Inheritance diagram for RMOL::HistoricalBookingHolder:



Public Member Functions

- const short [getNbOfFlights](#) () const
- const short [getNbOfUncensoredData](#) () const
- const stdair::NbOfBookings_T [getNbOfUncensoredBookings](#) () const
- const double [getUncensoredStandardDeviation](#) (const double &iMeanOfUncensoredBookings, const short iNbOfUncensoredData) const
- const double [getDemandMean](#) () const
- const double [getStandardDeviation](#) (const double) const
- const std::vector< bool > [getListOfToBeUnconstrainedFlags](#) () const
- const stdair::NbOfBookings_T & [getHistoricalBooking](#) (const short i) const
- const stdair::NbOfBookings_T & [getUnconstrainedDemand](#) (const short i) const
- const stdair::Flag_T & [getCensorshipFlag](#) (const short i) const
- const stdair::NbOfBookings_T & [getUnconstrainedDemandOnFirstElement](#) () const

- const stdair::NbOfBookings_T [calculateExpectedDemand](#) (const double, const double, const short, const stdair::NbOfBookings_T) const
- void [setUnconstrainedDemand](#) (const stdair::NbOfBookings_T &iExpectedDemand, const short i)
- void [addHistoricalBooking](#) (const [HistoricalBooking](#) &iHistoricalBooking)
- void [toStream](#) (std::ostream &ioOut) const
- const std::string [describe](#) () const
- void [display](#) () const
- virtual [~HistoricalBookingHolder](#) ()
- [HistoricalBookingHolder](#) ()

25.17.1 Detailed Description

Holder of a HistoricalBookingList object (for memory allocation and recollection purposes).

Definition at line 23 of file [HistoricalBookingHolder.hpp](#).

25.17.2 Constructor & Destructor Documentation

25.17.2.1 RMOL::HistoricalBookingHolder::~~HistoricalBookingHolder () [virtual]

Destructor.

Definition at line 23 of file [HistoricalBookingHolder.cpp](#).

25.17.2.2 RMOL::HistoricalBookingHolder::HistoricalBookingHolder ()

Constructor.

Protected to force the use of the Factory.

Definition at line 19 of file [HistoricalBookingHolder.cpp](#).

25.17.3 Member Function Documentation

25.17.3.1 const short RMOL::HistoricalBookingHolder::getNbOfFlights () const

Get number of flights.

Definition at line 28 of file [HistoricalBookingHolder.cpp](#).

Referenced by [RMOL::EMDetruncator::unconstrainUsingEMMethod\(\)](#), and [RMOL::Detruncator::unconstrainUsing-MultiplicativePickUp\(\)](#).

25.17.3.2 const short RMOL::HistoricalBookingHolder::getNbOfUncensoredData () const

Get number of uncensored booking data.

Definition at line 33 of file [HistoricalBookingHolder.cpp](#).

Referenced by [RMOL::EMDetruncator::unconstrainUsingEMMethod\(\)](#), and [RMOL::Detruncator::unconstrainUsing-MultiplicativePickUp\(\)](#).

25.17.3.3 const stdair::NbOfBookings_T RMOL::HistoricalBookingHolder::getNbOfUncensoredBookings () const

Get number of uncensored bookings.

Definition at line 49 of file [HistoricalBookingHolder.cpp](#).

References [RMOL::HistoricalBooking::getFlag\(\)](#), and [RMOL::HistoricalBooking::getNbOfBookings\(\)](#).

Referenced by [RMOL::EMDetruncator::unconstrainUsingEMMethod\(\)](#).

25.17.3.4 `const double RMOL::HistoricalBookingHolder::getUncensoredStandardDeviation (const double & iMeanOfUncensoredBookings, const short iNbOfUncensoredData) const`

Get standard deviation of uncensored bookings.

Definition at line 69 of file [HistoricalBookingHolder.cpp](#).

References [RMOL::HistoricalBooking::getNbOfBookings\(\)](#).

Referenced by [RMOL::EMDetruncator::unconstrainUsingEMMethod\(\)](#).

25.17.3.5 `const double RMOL::HistoricalBookingHolder::getDemandMean () const`

Get mean of historical demand.

Definition at line 95 of file [HistoricalBookingHolder.cpp](#).

References [RMOL::HistoricalBooking::getUnconstrainedDemand\(\)](#).

Referenced by [RMOL::EMDetruncator::unconstrainUsingEMMethod\(\)](#).

25.17.3.6 `const double RMOL::HistoricalBookingHolder::getStandardDeviation (const double iDemandMean) const`

Get standard deviation of demand.

Definition at line 116 of file [HistoricalBookingHolder.cpp](#).

References [RMOL::HistoricalBooking::getUnconstrainedDemand\(\)](#).

Referenced by [RMOL::EMDetruncator::unconstrainUsingEMMethod\(\)](#).

25.17.3.7 `const std::vector< bool > RMOL::HistoricalBookingHolder::getListOfToBeUnconstrainedFlags () const`

Get the list of flags of need to be unconstrained.

Definition at line 140 of file [HistoricalBookingHolder.cpp](#).

References [RMOL::HistoricalBooking::getFlag\(\)](#).

Referenced by [RMOL::EMDetruncator::unconstrainUsingEMMethod\(\)](#).

25.17.3.8 `const stdair::NbOfBookings_T & RMOL::HistoricalBookingHolder::getHistoricalBooking (const short i) const`

Get the historical booking of the (i+1)-th flight.

Definition at line 161 of file [HistoricalBookingHolder.cpp](#).

References [RMOL::HistoricalBooking::getNbOfBookings\(\)](#).

Referenced by [RMOL::Detruncator::unconstrainUsingMultiplicativePickUp\(\)](#).

25.17.3.9 `const stdair::NbOfBookings_T & RMOL::HistoricalBookingHolder::getUnconstrainedDemand (const short i) const`

Get the unconstraining demand of the (i+1)-th flight.

Definition at line 169 of file [HistoricalBookingHolder.cpp](#).

References [RMOL::HistoricalBooking::getUnconstrainedDemand\(\)](#).

Referenced by [getUnconstrainedDemandOnFirstElement\(\)](#), and [RMOL::EMDetruncator::unconstrainUsingEMMethod\(\)](#).

25.17.3.10 `const stdair::Flag_T & RMOL::HistoricalBookingHolder::getCensorshipFlag (const short i) const`

Get the flag of the (i+1)-th flight.

Definition at line 177 of file [HistoricalBookingHolder.cpp](#).

References [RMOL::HistoricalBooking::getFlag\(\)](#).

Referenced by [RMOL::Detruncator::unconstrainUsingMultiplicativePickUp\(\)](#).

25.17.3.11 `const stdair::NbOfBookings_T& RMOL::HistoricalBookingHolder::getUnconstrainedDemandOnFirstElement ()`
`const [inline]`

Get the unconstraining demand of the first flight.

Definition at line 60 of file [HistoricalBookingHolder.hpp](#).

References [getUnconstrainedDemand\(\)](#).

25.17.3.12 `const stdair::NbOfBookings_T RMOL::HistoricalBookingHolder::calculateExpectedDemand (const double iMean, const double iSD, const short i, const stdair::NbOfBookings_T iDemand) const`

Calculate the expected demand.

Definition at line 191 of file [HistoricalBookingHolder.cpp](#).

References [RMOL::HistoricalBooking::getNbOfBookings\(\)](#).

25.17.3.13 `void RMOL::HistoricalBookingHolder::setUnconstrainedDemand (const stdair::NbOfBookings_T & iExpectedDemand, const short i)`

Set the expected historical demand of the (i+1)-th flight.

Definition at line 185 of file [HistoricalBookingHolder.cpp](#).

Referenced by [RMOL::EMDetruncator::unconstrainUsingEMMethod\(\)](#), and [RMOL::Detruncator::unconstrainUsing-MultiplicativePickUp\(\)](#).

25.17.3.14 `void RMOL::HistoricalBookingHolder::addHistoricalBooking (const HistoricalBooking & iHistoricalBooking)`

Add a [HistoricalBooking](#) object to the holder.

Definition at line 236 of file [HistoricalBookingHolder.cpp](#).

25.17.3.15 `void RMOL::HistoricalBookingHolder::toStream (std::ostream & ioOut) const`

Dump a Business Object into an output stream.

Parameters

<i>ostream&</i>	the output stream
---------------------	-------------------

Returns

ostream& the output stream.

Definition at line 241 of file [HistoricalBookingHolder.cpp](#).

References [RMOL::HistoricalBooking::getFlag\(\)](#), [RMOL::HistoricalBooking::getNbOfBookings\(\)](#), and [RMOL::HistoricalBooking::getUnconstrainedDemand\(\)](#).

Referenced by [display\(\)](#).

25.17.3.16 `const std::string RMOL::HistoricalBookingHolder::describe () const`

Give a description of the structure (for display purposes).

Definition at line 265 of file [HistoricalBookingHolder.cpp](#).

25.17.3.17 `void RMOL::HistoricalBookingHolder::display () const`

Display on standard output.

Definition at line 273 of file [HistoricalBookingHolder.cpp](#).

References [toStream\(\)](#).

The documentation for this struct was generated from the following files:

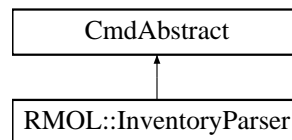
- [rmol/bom/HistoricalBookingHolder.hpp](#)
- [rmol/bom/HistoricalBookingHolder.cpp](#)

25.18 RMOL::InventoryParser Class Reference

Class filling the virtual class list (representing a list of classes/buckets) from a given input inventory.

```
#include <rmol/command/InventoryParser.hpp>
```

Inheritance diagram for RMOL::InventoryParser:



Static Public Member Functions

- static bool [parseInputFileAndBuildBom](#) (const std::string &inputFileName, stdair::BomRoot &)

25.18.1 Detailed Description

Class filling the virtual class list (representing a list of classes/buckets) from a given input inventory.

Definition at line 25 of file [InventoryParser.hpp](#).

25.18.2 Member Function Documentation

25.18.2.1 bool RMOL::InventoryParser::parseInputFileAndBuildBom (const std::string & *inputFileName*, stdair::BomRoot & *ioBomRoot*) [static]

Parse the input values from a CSV-formatted inventory file.

Parameters

<i>const</i>	std::string& inputFileName Inventory file to be parsed.
<i>stdair::Bom-Root&</i>	The BOM tree.

Returns

bool Whether or not the parsing was successful.

Definition at line 36 of file [InventoryParser.cpp](#).

Referenced by [RMOL::RMOL_Service::parseAndLoad\(\)](#).

The documentation for this class was generated from the following files:

- [rmol/command/InventoryParser.hpp](#)
- [rmol/command/InventoryParser.cpp](#)

25.19 RMOL::MCOptimiser Class Reference

```
#include <rmol/bom/MCOptimiser.hpp>
```

Static Public Member Functions

- static void [optimalOptimisationByMCIntegration](#) (stdair::LegCabin &)
- static
stdair::GeneratedDemandVector_T [generateDemandVector](#) (const stdair::MeanValue_T &, const stdair::StdDevValue_T &, const unsigned int &)
- static void [optimisationByMCIntegration](#) (stdair::LegCabin &)

25.19.1 Detailed Description

Utility methods for the Monte-Carlo algorithms.

Definition at line 19 of file [MCOptimiser.hpp](#).

25.19.2 Member Function Documentation

25.19.2.1 void RMOL::MCOptimiser::optimalOptimisationByMCIntegration (stdair::LegCabin & *ioLegCabin*) [static]

Calculate the optimal protections for the set of buckets/classes given in input, and update those buckets accordingly.

The Monte Carlo Integration algorithm (see The Theory and Practice of Revenue Management, by Kalyan T. Talluri and Garret J. van Ryzin, Kluwer Academic Publishers, for the details) is used.

Definition at line 28 of file [MCOptimiser.cpp](#).

25.19.2.2 stdair::GeneratedDemandVector_T RMOL::MCOptimiser::generateDemandVector (const stdair::MeanValue_T & *iMean*, const stdair::StdDevValue_T & *iStdDev*, const unsigned int & *K*) [static]

Monte-Carlo

Definition at line 154 of file [MCOptimiser.cpp](#).

Referenced by [optimisationByMCIntegration\(\)](#).

25.19.2.3 void RMOL::MCOptimiser::optimisationByMCIntegration (stdair::LegCabin & *ioLegCabin*) [static]

Definition at line 175 of file [MCOptimiser.cpp](#).

References [generateDemandVector\(\)](#).

Referenced by [RMOL::Optimiser::optimiseUsingOnDForecast\(\)](#).

The documentation for this class was generated from the following files:

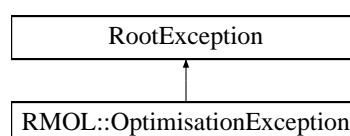
- [rmol/bom/MCOptimiser.hpp](#)
- [rmol/bom/MCOptimiser.cpp](#)

25.20 RMOL::OptimisationException Class Reference

Optimisation-related exception.

```
#include <rmol/RMOL_Types.hpp>
```

Inheritance diagram for RMOL::OptimisationException:



Public Member Functions

- [OptimisationException](#) (const std::string &iWhat)

25.20.1 Detailed Description

Optimisation-related exception.

Definition at line 61 of file [RMOL_Types.hpp](#).

25.20.2 Constructor & Destructor Documentation

25.20.2.1 RMOL::OptimisationException::OptimisationException (const std::string & iWhat) [inline]

Constructor.

Definition at line 64 of file [RMOL_Types.hpp](#).

The documentation for this class was generated from the following file:

- [rmol/RMOL_Types.hpp](#)

25.21 RMOL::Optimiser Class Reference

```
#include <rmol/command/Optimiser.hpp>
```

Static Public Member Functions

- static void [optimalOptimisationByMCIntegration](#) (const int K, stdair::LegCabin &)
- static void [optimalOptimisationByDP](#) (stdair::LegCabin &)
- static void [heuristicOptimisationByEmsr](#) (stdair::LegCabin &)
- static void [heuristicOptimisationByEmsrA](#) (stdair::LegCabin &)
- static void [heuristicOptimisationByEmsrB](#) (stdair::LegCabin &)
- static void [optimise](#) (stdair::FlightDate &)
- static void [buildVirtualClassListForLegBasedOptimisation](#) (stdair::LegCabin &)
- static double [optimiseUsingOnDForecast](#) (stdair::FlightDate &, const bool &iReduceFluctuations=false)

25.21.1 Detailed Description

Class wrapping the optimisation algorithms.

Definition at line 18 of file [Optimiser.hpp](#).

25.21.2 Member Function Documentation

25.21.2.1 void RMOL::Optimiser::optimalOptimisationByMCIntegration (const int K, stdair::LegCabin & ioLegCabin) [static]

Monte Carlo Integration algorithm.

Calculate the optimal protections for the set of buckets/classes given in input, and update those buckets accordingly.

The Monte Carlo Integration algorithm (see The Theory and Practice of Revenue Management, by Kalyan T. Talluri and Garret J. van Ryzin, Kluwer Academic Publishers, for the details) is used. Hence, K is the number of random draws to perform. 100 is a minimum for K, as statistics must be drawn from those random generations.

Definition at line 29 of file [Optimiser.cpp](#).

Referenced by [optimise\(\)](#).

25.21.2.2 void RMOL::Optimiser::optimalOptimisationByDP (stdair::LegCabin & *ioLegCabin*) [static]

Dynamic Programming.

Definition at line 63 of file [Optimiser.cpp](#).

25.21.2.3 void RMOL::Optimiser::heuristicOptimisationByEmsr (stdair::LegCabin & *ioLegCabin*) [static]

EMRS algorithm.

Definition at line 68 of file [Optimiser.cpp](#).

25.21.2.4 void RMOL::Optimiser::heuristicOptimisationByEmsrA (stdair::LegCabin & *ioLegCabin*) [static]

EMRS-a algorithm.

Definition at line 73 of file [Optimiser.cpp](#).

25.21.2.5 void RMOL::Optimiser::heuristicOptimisationByEmsrB (stdair::LegCabin & *ioLegCabin*) [static]

EMRS-b algorithm.

Definition at line 78 of file [Optimiser.cpp](#).

25.21.2.6 void RMOL::Optimiser::optimise (stdair::FlightDate & *ioFlightDate*) [static]

Optimise a flight-date using leg-based Monte Carlo Integration.

Definition at line 83 of file [Optimiser.cpp](#).

References [buildVirtualClassListForLegBasedOptimisation\(\)](#), and [optimalOptimisationByMCIntegration\(\)](#).

25.21.2.7 void RMOL::Optimiser::buildVirtualClassListForLegBasedOptimisation (stdair::LegCabin & *ioLegCabin*) [static]

Build the virtual class list for the given leg-cabin.

Definition at line 112 of file [Optimiser.cpp](#).

Referenced by [optimise\(\)](#).

25.21.2.8 double RMOL::Optimiser::optimiseUsingOnDForecast (stdair::FlightDate & *ioFlightDate*, const bool & *iReduceFluctuations* = false) [static]

Optimiser

Definition at line 156 of file [Optimiser.cpp](#).

References [RMOL::MCOptimiser::optimisationByMCIntegration\(\)](#).

Referenced by [RMOL::RMOL_Service::optimiseOnD\(\)](#), [RMOL::RMOL_Service::optimiseOnDUsingAdvancedRM-Cooperation\(\)](#), and [RMOL::RMOL_Service::optimiseOnDUsingRMCooperation\(\)](#).

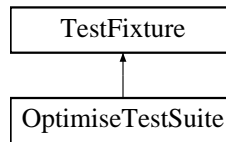
The documentation for this class was generated from the following files:

- [rmol/command/Optimiser.hpp](#)
- [rmol/command/Optimiser.cpp](#)

25.22 OptimiseTestSuite Class Reference

```
#include <test/rmol/OptimiseTestSuite.hpp>
```

Inheritance diagram for OptimiseTestSuite:



Public Member Functions

- void [testOptimiseMC](#) ()
- void [testOptimiseDP](#) ()
- void [testOptimiseEMSR](#) ()
- void [testOptimiseEMSRa](#) ()
- void [testOptimiseEMSRb](#) ()
- [OptimiseTestSuite](#) ()

Protected Attributes

- std::stringstream [_describeKey](#)

25.22.1 Detailed Description

Definition at line 6 of file [OptimiseTestSuite.hpp](#).

25.22.2 Constructor & Destructor Documentation

25.22.2.1 OptimiseTestSuite::OptimiseTestSuite ()

Test some error detection functionalities. Constructor.

25.22.3 Member Function Documentation

25.22.3.1 void OptimiseTestSuite::testOptimiseMC ()

Test the Monte-Carlo (MC) Optimisation functionality.

25.22.3.2 void OptimiseTestSuite::testOptimiseDP ()

Test the Dynamic Programming (DP) Optimisation functionality.

25.22.3.3 void OptimiseTestSuite::testOptimiseEMSR ()

Test the Expected Marginal Seat Revenue (EMSR) Optimisation functionality.

25.22.3.4 void OptimiseTestSuite::testOptimiseEMSRa ()

Test the Expected Marginal Seat Revenue, variant a (EMSR-a), Optimisation functionality.

25.22.3.5 void OptimiseTestSuite::testOptimiseEMSRb ()

Test the Expected Marginal Seat Revenue, variant b (EMSR-b), Optimisation functionality.

25.22.4 Member Data Documentation

25.22.4.1 std::stringstream OptimiseTestSuite::describeKey [protected]

Definition at line 43 of file [OptimiseTestSuite.hpp](#).

The documentation for this class was generated from the following file:

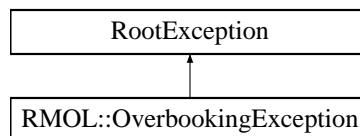
- [test/rmol/OptimiseTestSuite.hpp](#)

25.23 RMOL::OverbookingException Class Reference

Overbooking-related exception.

```
#include <rmol/RMOL_Types.hpp>
```

Inheritance diagram for RMOL::OverbookingException:



Public Member Functions

- [OverbookingException](#) (const std::string &iWhat)

25.23.1 Detailed Description

Overbooking-related exception.

Definition at line 31 of file [RMOL_Types.hpp](#).

25.23.2 Constructor & Destructor Documentation

25.23.2.1 RMOL::OverbookingException::OverbookingException (const std::string & iWhat) [inline]

Constructor.

Definition at line 34 of file [RMOL_Types.hpp](#).

The documentation for this class was generated from the following file:

- [rmol/RMOL_Types.hpp](#)

25.24 RMOL::RMOL_Service Class Reference

Interface for the [RMOL](#) Services.

```
#include <rmol/RMOL_Service.hpp>
```

Public Member Functions

- [RMOL_Service](#) (const stdair::BasLogParams &, const stdair::BasDBParams &)
- [RMOL_Service](#) (const stdair::BasLogParams &)
- [RMOL_Service](#) (stdair::STDAIR_ServicePtr_T)
- void [parseAndLoad](#) (const stdair::CabinCapacity_T &iCabinCapacity, const stdair::Filename_T &iDemand-AndClassDataFile)

- void [setUpStudyStatManager](#) ()
- [~RMOL_Service](#) ()
- void [buildSampleBom](#) ()
- void [optimalOptimisationByMCIntegration](#) (const int K)
- void [optimalOptimisationByDP](#) ()
- void [heuristicOptimisationByEmsr](#) ()
- void [heuristicOptimisationByEmsrA](#) ()
- void [heuristicOptimisationByEmsrB](#) ()
- bool [optimise](#) (stdair::FlightDate &, const stdair::DateTime_T &, const stdair::ForecastingMethod &, const stdair::PartnershipTechnique &)
- void [forecastOnD](#) (const stdair::DateTime_T &)
- stdair::YieldFeatures * [getYieldFeatures](#) (const stdair::OnDDate &, const stdair::CabinCode_T &, stdair::BomRoot &)
- void [forecastOnD](#) (const stdair::YieldFeatures &, stdair::OnDDate &, const stdair::CabinCode_T &, const stdair::DTD_T &, stdair::BomRoot &)
- void [setOnDForecast](#) (const stdair::AirlineClassList &, const stdair::MeanValue_T &, const stdair::StdDevValue_T &, stdair::OnDDate &, const stdair::CabinCode_T &, stdair::BomRoot &)
- void [setOnDForecast](#) (const stdair::AirlineCode_T &, const stdair::Date_T &, const stdair::AirportCode_T &, const stdair::AirportCode_T &, const stdair::CabinCode_T &, const stdair::ClassCode_T &, const stdair::MeanValue_T &, const stdair::StdDevValue_T &, const stdair::Yield_T &, stdair::BomRoot &)
- void [setOnDForecast](#) (const stdair::AirlineCodeList_T &, const stdair::AirlineCode_T &, const stdair::Date_T &, const stdair::AirportCode_T &, const stdair::AirportCode_T &, const stdair::CabinCode_T &, const stdair::ClassCodeList_T &, const stdair::MeanValue_T &, const stdair::StdDevValue_T &, const stdair::Yield_T &, stdair::BomRoot &)
- void [resetDemandInformation](#) (const stdair::DateTime_T &)
- void [resetDemandInformation](#) (const stdair::DateTime_T &, const stdair::Inventory &)
- void [projectAggregatedDemandOnLegCabins](#) (const stdair::DateTime_T &)
- void [projectOnDDemandOnLegCabinsUsingYP](#) (const stdair::DateTime_T &)
- void [projectOnDDemandOnLegCabinsUsingDA](#) (const stdair::DateTime_T &)
- void [projectOnDDemandOnLegCabinsUsingDYP](#) (const stdair::DateTime_T &)
- void [projectOnDDemandOnLegCabinsUsingDYP](#) (const stdair::DateTime_T &, const stdair::Inventory &)
- void [optimiseOnD](#) (const stdair::DateTime_T &)
- void [optimiseOnDUsingRMCooperation](#) (const stdair::DateTime_T &)
- void [optimiseOnDUsingAdvancedRMCooperation](#) (const stdair::DateTime_T &)
- void [updateBidPrice](#) (const stdair::DateTime_T &)
- void [updateBidPrice](#) (const stdair::FlightDate &, stdair::BomRoot &)
- std::string [jsonExport](#) (const stdair::AirlineCode_T &, const stdair::FlightNumber_T &, const stdair::Date_T & iDepartureDate) const
- std::string [csvDisplay](#) () const

25.24.1 Detailed Description

Interface for the [RMOL](#) Services.

Definition at line 39 of file [RMOL_Service.hpp](#).

25.24.2 Constructor & Destructor Documentation

25.24.2.1 RMOL::RMOL_Service::RMOL_Service (const stdair::BasLogParams & *iLogParams*, const stdair::BasDBParams & *iDBParams*)

Constructor.

The `initRmolService()` method is called; see the corresponding documentation for more details.

A reference on an output stream is given, so that log outputs can be directed onto that stream.

Moreover, database connection parameters are given, so that a session can be created on the corresponding database.

Parameters

<i>const</i>	stdair::BasLogParams& Parameters for the output log stream.
<i>const</i>	stdair::BasDBParams& Parameters for the database access.

Definition at line 84 of file [RMOL_Service.cpp](#).

25.24.2.2 RMOL::RMOL_Service::RMOL_Service (const stdair::BasLogParams & iLogParams)

Constructor.

The initRmolService() method is called; see the corresponding documentation for more details.

Moreover, a reference on an output stream is given, so that log outputs can be directed onto that stream.

Parameters

<i>const</i>	stdair::BasLogParams& Parameters for the output log stream.
--------------	---

Definition at line 63 of file [RMOL_Service.cpp](#).

25.24.2.3 RMOL::RMOL_Service::RMOL_Service (stdair::STDAIR_ServicePtr_T ioSTDAIRServicePtr)

Constructor.

The initRmolService() method is called; see the corresponding documentation for more details.

Moreover, as no reference on any output stream is given, it is assumed that the StdAir log service has already been initialised with the proper log output stream by some other methods in the calling chain (for instance, when the [RMOL_Service](#) is itself being initialised by another library service such as [AIRINV_Service](#)).

Parameters

<i>STDAIR_ServicePtr_T</i>	the shared pointer of stdair service.
----------------------------	---------------------------------------

Definition at line 106 of file [RMOL_Service.cpp](#).

25.24.2.4 RMOL::RMOL_Service::~~RMOL_Service ()

Destructor.

Definition at line 123 of file [RMOL_Service.cpp](#).

25.24.3 Member Function Documentation

25.24.3.1 void RMOL::RMOL_Service::parseAndLoad (const stdair::CabinCapacity_T & iCabinCapacity, const stdair::Filename_T & iDemandAndClassDataFile)

Parse the optimisation-related data and load them into memory.

First, the [STDAIR_Service::buildDummyInventory\(\)](#) method is called, for [RMOL](#) and with the given cabin capacity, in order to build the minimum required flight-date structure in order to perform an optimisation on a leg-cabin.

The CSV input file describes the problem to be optimised, i.e.:

- the demand specifications for all the booking classes (mean and standard deviations for the demand distribution); the yields corresponding to those booking classes.

That CSV file is parsed and instantiated in memory accordingly. The leg-cabin capacity has been set at the initialisation of the ([RMOL](#)) service.

Parameters

<i>const</i>	stdair::CabinCapacity& Capacity of the leg-cabin to be optimised.
<i>const</i>	stdair::Filename_T& (CSV) input file.

Definition at line 200 of file [RMOL_Service.cpp](#).

References [RMOL::InventoryParser::parseInputFileAndBuildBom\(\)](#).

Referenced by [main\(\)](#).

25.24.3.2 void RMOL::RMOL_Service::setUpStudyStatManager ()

Set up the StudyStatManager.

25.24.3.3 void RMOL::RMOL_Service::buildSampleBom ()

Build a sample BOM tree, and attach it to the BomRoot instance.

See Also

[stdair::CmdBomManager::buildSampleBom\(\)](#) for more details.

Definition at line 224 of file [RMOL_Service.cpp](#).

Referenced by [main\(\)](#).

25.24.3.4 void RMOL::RMOL_Service::optimalOptimisationByMCIntegration (const int K)

Single resource optimization using the Monte Carlo algorithm.

Definition at line 272 of file [RMOL_Service.cpp](#).

Referenced by [optimise\(\)](#).

25.24.3.5 void RMOL::RMOL_Service::optimalOptimisationByDP ()

Single resource optimization using dynamic programming.

Definition at line 312 of file [RMOL_Service.cpp](#).

Referenced by [optimise\(\)](#).

25.24.3.6 void RMOL::RMOL_Service::heuristicOptimisationByEmsr ()

Single resource optimization using EMSR heuristic.

Definition at line 316 of file [RMOL_Service.cpp](#).

Referenced by [optimise\(\)](#).

25.24.3.7 void RMOL::RMOL_Service::heuristicOptimisationByEmsrA ()

Single resource optimization using EMSR-a heuristic.

Definition at line 357 of file [RMOL_Service.cpp](#).

Referenced by [optimise\(\)](#).

25.24.3.8 void RMOL::RMOL_Service::heuristicOptimisationByEmsrB ()

Single resource optimization using EMSR-b heuristic.

Definition at line 378 of file [RMOL_Service.cpp](#).

Referenced by [optimise\(\)](#).

25.24.3.9 `bool RMOL::RMOL_Service::optimise (stdair::FlightDate & ioFlightDate, const stdair::DateTime_T & iRMEventTime, const stdair::ForecastingMethod & iForecastingMethod, const stdair::PartnershipTechnique & iPartnershipTechnique)`

Optimise (revenue management) an flight-date/network-date

Definition at line 399 of file [RMOL_Service.cpp](#).

References [forecastOnD\(\)](#), [RMOL::Forecaster::forecastUsingAdditivePickUp\(\)](#), [RMOL::Forecaster::forecastUsingMultiplicativePickUp\(\)](#), [optimiseOnD\(\)](#), [optimiseOnDUsingAdvancedRMCooperation\(\)](#), [optimiseOnDUsingRMCooperation\(\)](#), [projectAggregatedDemandOnLegCabins\(\)](#), [projectOnDDemandOnLegCabinsUsingDYP\(\)](#), [projectOnDDemandOnLegCabinsUsingYP\(\)](#), [resetDemandInformation\(\)](#), and [updateBidPrice\(\)](#).

25.24.3.10 `void RMOL::RMOL_Service::forecastOnD (const stdair::DateTime_T & iRMEventTime)`

[Forecaster](#)

Definition at line 500 of file [RMOL_Service.cpp](#).

References [RMOL::DEFAULT_DCP_LIST](#), and [getYieldFeatures\(\)](#).

Referenced by [optimise\(\)](#).

25.24.3.11 `stdair::YieldFeatures * RMOL::RMOL_Service::getYieldFeatures (const stdair::OnDDate & iOnDDate, const stdair::CabinCode_T & iCabinCode, stdair::BomRoot & iBomRoot)`

Definition at line 573 of file [RMOL_Service.cpp](#).

Referenced by [forecastOnD\(\)](#).

25.24.3.12 `void RMOL::RMOL_Service::forecastOnD (const stdair::YieldFeatures & iYieldFeatures, stdair::OnDDate & iOnDDate, const stdair::CabinCode_T & iCabinCode, const stdair::DTD_T & iDTD, stdair::BomRoot & iBomRoot)`

Definition at line 646 of file [RMOL_Service.cpp](#).

References [setOnDForecast\(\)](#).

25.24.3.13 `void RMOL::RMOL_Service::setOnDForecast (const stdair::AirlineClassList & iAirlineClassList, const stdair::MeanValue_T & iMeanValue, const stdair::StdDevValue_T & iStdDevValue, stdair::OnDDate & iOnDDate, const stdair::CabinCode_T & iCabinCode, stdair::BomRoot & iBomRoot)`

Definition at line 761 of file [RMOL_Service.cpp](#).

Referenced by [forecastOnD\(\)](#).

25.24.3.14 `void RMOL::RMOL_Service::setOnDForecast (const stdair::AirlineCode_T & iAirlineCode, const stdair::Date_T & iDepartureDate, const stdair::AirportCode_T & iOrigin, const stdair::AirportCode_T & iDestination, const stdair::CabinCode_T & iCabinCode, const stdair::ClassCode_T & iClassCode, const stdair::MeanValue_T & iMeanValue, const stdair::StdDevValue_T & iStdDevValue, const stdair::Yield_T & iYield, stdair::BomRoot & iBomRoot)`

Definition at line 820 of file [RMOL_Service.cpp](#).

25.24.3.15 `void RMOL::RMOL_Service::setOnDForecast (const stdair::AirlineCodeList_T & iAirlineCodeList, const stdair::AirlineCode_T & iAirlineCode, const stdair::Date_T & iDepartureDate, const stdair::AirportCode_T & iOrigin, const stdair::AirportCode_T & iDestination, const stdair::CabinCode_T & iCabinCode, const stdair::ClassCodeList_T & iClassCodeList, const stdair::MeanValue_T & iMeanValue, const stdair::StdDevValue_T & iStdDevValue, const stdair::Yield_T & iYield, stdair::BomRoot & iBomRoot)`

Definition at line 882 of file [RMOL_Service.cpp](#).

25.24.3.16 `void RMOL::RMOL_Service::resetDemandInformation (const stdair::DateTime_T & iRMEventTime)`

Definition at line 997 of file [RMOL_Service.cpp](#).

Referenced by [optimise\(\)](#), [optimiseOnDUsingAdvancedRMCooperation\(\)](#), and [optimiseOnDUsingRMCooperation\(\)](#).

25.24.3.17 void RMOL::RMOL_Service::resetDemandInformation (const stdair::DateTime_T & *iRMEventTime*, const stdair::Inventory & *ilInventory*)

Definition at line 1023 of file [RMOL_Service.cpp](#).

References [RMOL::DEFAULT_DCP_LIST](#).

25.24.3.18 void RMOL::RMOL_Service::projectAggregatedDemandOnLegCabins (const stdair::DateTime_T & *iRMEventTime*)

Definition at line 1071 of file [RMOL_Service.cpp](#).

References [RMOL::DEFAULT_DCP_LIST](#).

Referenced by [optimise\(\)](#).

25.24.3.19 void RMOL::RMOL_Service::projectOnDDemandOnLegCabinsUsingYP (const stdair::DateTime_T & *iRMEventTime*)

Definition at line 1176 of file [RMOL_Service.cpp](#).

References [RMOL::DEFAULT_DCP_LIST](#).

Referenced by [optimise\(\)](#).

25.24.3.20 void RMOL::RMOL_Service::projectOnDDemandOnLegCabinsUsingDA (const stdair::DateTime_T & *iRMEventTime*)

Definition at line 1451 of file [RMOL_Service.cpp](#).

References [RMOL::DEFAULT_DCP_LIST](#).

25.24.3.21 void RMOL::RMOL_Service::projectOnDDemandOnLegCabinsUsingDYP (const stdair::DateTime_T & *iRMEventTime*)

Definition at line 1607 of file [RMOL_Service.cpp](#).

Referenced by [optimise\(\)](#), [optimiseOnDUsingAdvancedRMCooperation\(\)](#), and [optimiseOnDUsingRMCooperation\(\)](#).

25.24.3.22 void RMOL::RMOL_Service::projectOnDDemandOnLegCabinsUsingDYP (const stdair::DateTime_T & *iRMEventTime*, const stdair::Inventory & *ilInventory*)

Definition at line 1633 of file [RMOL_Service.cpp](#).

References [RMOL::DEFAULT_DCP_LIST](#).

25.24.3.23 void RMOL::RMOL_Service::optimiseOnD (const stdair::DateTime_T & *iRMEventTime*)

[Optimiser](#)

Definition at line 1275 of file [RMOL_Service.cpp](#).

References [RMOL::DEFAULT_DCP_LIST](#), and [RMOL::Optimiser::optimiseUsingOnDForecast\(\)](#).

Referenced by [optimise\(\)](#).

25.24.3.24 void RMOL::RMOL_Service::optimiseOnDUsingRMCooperation (const stdair::DateTime_T & *iRMEventTime*)

Definition at line 1749 of file [RMOL_Service.cpp](#).

References [RMOL::DEFAULT_DCP_LIST](#), [RMOL::Optimiser::optimiseUsingOnDForecast\(\)](#), [projectOnDDemandOnLegCabinsUsingDYP\(\)](#), and [resetDemandInformation\(\)](#).

Referenced by [optimise\(\)](#).

25.24.3.25 void RMOL::RMOL_Service::optimiseOnDUsingAdvancedRMCooperation (const stdair::DateTime_T & *iRMEventTime*)

Definition at line 1809 of file [RMOL_Service.cpp](#).

References [RMOL::DEFAULT_DCP_LIST](#), [RMOL::Optimiser::optimiseUsingOnDForecast\(\)](#), [projectOnDDemand-OnLegCabinsUsingDYP\(\)](#), [resetDemandInformation\(\)](#), and [updateBidPrice\(\)](#).

Referenced by [optimise\(\)](#).

25.24.3.26 void RMOL::RMOL_Service::updateBidPrice (const stdair::DateTime_T & *iRMEventTime*)

Definition at line 1324 of file [RMOL_Service.cpp](#).

References [RMOL::DEFAULT_DCP_LIST](#).

Referenced by [optimise\(\)](#), and [optimiseOnDUsingAdvancedRMCooperation\(\)](#).

25.24.3.27 void RMOL::RMOL_Service::updateBidPrice (const stdair::FlightDate & *iFlightDate*, stdair::BomRoot & *iBomRoot*)

Definition at line 1372 of file [RMOL_Service.cpp](#).

25.24.3.28 std::string RMOL::RMOL_Service::jsonExport (const stdair::AirlineCode_T & , const stdair::FlightNumber_T & , const stdair::Date_T & *iDepartureDate*) const

Recursively dump, in the returned string and in JSON format, the flight-date corresponding to the parameters given as input.

Parameters

<i>const</i>	stdair::AirlineCode_T& Airline code of the flight to dump.
<i>const</i>	stdair::FlightNumber_T& Flight number of the flight to dump.
<i>const</i>	stdair::Date_T& Departure date of a flight to dump.

Returns

std::string Output string in which the BOM tree is JSON-ified.

25.24.3.29 std::string RMOL::RMOL_Service::csvDisplay () const

Recursively display (dump in the returned string) the objects of the BOM tree.

Returns

std::string Output string in which the BOM tree is logged/dumped.

The documentation for this class was generated from the following files:

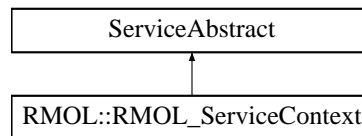
- [rmol/RMOL_Service.hpp](#)
- [rmol/service/RMOL_Service.cpp](#)

25.25 RMOL::RMOL_ServiceContext Class Reference

Inner class holding the context for the [RMOL](#) Service object.

```
#include <rmol/service/RMOL_ServiceContext.hpp>
```

Inheritance diagram for RMOL::RMOL_ServiceContext:



Friends

- class [RMOL_Service](#)
- class [FacRmolServiceContext](#)

25.25.1 Detailed Description

Inner class holding the context for the [RMOL](#) Service object.

Definition at line 29 of file [RMOL_ServiceContext.hpp](#).

25.25.2 Friends And Related Function Documentation

25.25.2.1 friend class [RMOL_Service](#) [friend]

The [RMOL_Service](#) class should be the sole class to get access to ServiceContext content: general users do not want to bother with a context interface.

Definition at line 35 of file [RMOL_ServiceContext.hpp](#).

25.25.2.2 friend class [FacRmolServiceContext](#) [friend]

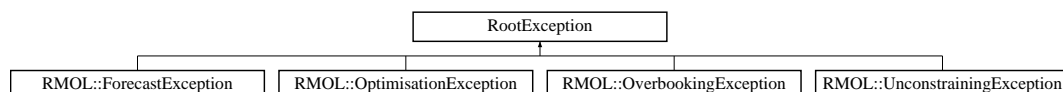
Definition at line 36 of file [RMOL_ServiceContext.hpp](#).

The documentation for this class was generated from the following files:

- [rmol/service/RMOL_ServiceContext.hpp](#)
- [rmol/service/RMOL_ServiceContext.cpp](#)

25.26 RootException Class Reference

Inheritance diagram for RootException:

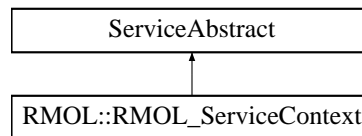


The documentation for this class was generated from the following file:

- [rmol/RMOL_Types.hpp](#)

25.27 ServiceAbstract Class Reference

Inheritance diagram for ServiceAbstract:

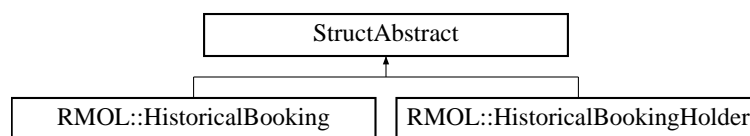


The documentation for this class was generated from the following file:

- [rmol/service/RMOL_ServiceContext.hpp](#)

25.28 StructAbstract Class Reference

Inheritance diagram for StructAbstract:

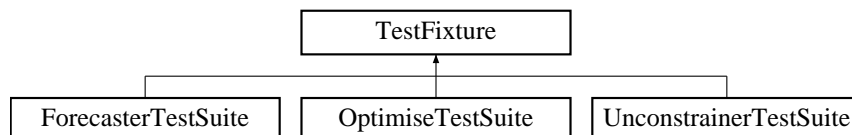


The documentation for this class was generated from the following file:

- [rmol/bom/HistoricalBooking.hpp](#)

25.29 TestFixture Class Reference

Inheritance diagram for TestFixture:



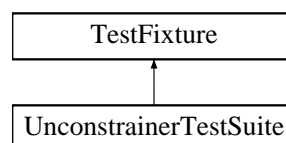
The documentation for this class was generated from the following file:

- [test/rmol/OptimiseTestSuite.hpp](#)

25.30 UnconstrainerTestSuite Class Reference

```
#include <test/rmol/UnconstrainerTestSuite.hpp>
```

Inheritance diagram for UnconstrainerTestSuite:



Public Member Functions

- void [testUnconstrainingByEM](#) ()
- [UnconstrainerTestSuite](#) ()

Protected Attributes

- std::stringstream [_describeKey](#)

25.30.1 Detailed Description

Definition at line 6 of file [UnconstrainerTestSuite.hpp](#).

25.30.2 Constructor & Destructor Documentation

25.30.2.1 UnconstrainerTestSuite::UnconstrainerTestSuite ()

Constructor.

25.30.3 Member Function Documentation

25.30.3.1 void UnconstrainerTestSuite::testUnconstrainingByEM ()

Test data unconstraining by Expectation Maximization.

25.30.4 Member Data Documentation

25.30.4.1 std::stringstream UnconstrainerTestSuite::_describeKey [protected]

Definition at line 19 of file [UnconstrainerTestSuite.hpp](#).

The documentation for this class was generated from the following file:

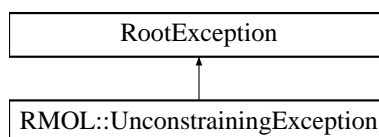
- test/rmol/[UnconstrainerTestSuite.hpp](#)

25.31 RMOL::UnconstrainingException Class Reference

Unconstraining-related exception.

```
#include <rmol/RMOL_Types.hpp>
```

Inheritance diagram for RMOL::UnconstrainingException:



Public Member Functions

- [UnconstrainingException](#) (const std::string &iWhat)

25.31.1 Detailed Description

Unconstraining-related exception.

Definition at line 41 of file [RMOL_Types.hpp](#).

25.31.2 Constructor & Destructor Documentation

25.31.2.1 RMOL::UnconstrainingException::UnconstrainingException (const std::string & *iWhat*) [inline]

Constructor.

Definition at line 44 of file [RMOL_Types.hpp](#).

The documentation for this class was generated from the following file:

- [rmol/RMOL_Types.hpp](#)

25.32 RMOL::Utilities Class Reference

```
#include <rmol/bom/Utilities.hpp>
```

Static Public Member Functions

- static void [computeDistributionParameters](#) (const [UnconstrainedDemandVector_T](#) &, double &, double &)
- static stdair::DCPList_T [buildRemainingDCPList](#) (const stdair::DTD_T &)
- static stdair::DCPList_T [buildRemainingDCPList2](#) (const stdair::DTD_T &)
- static stdair::NbOfSegments_T [getNbOfDepartedSimilarSegments](#) (const stdair::SegmentCabin &, const stdair::Date_T &)

25.32.1 Detailed Description

Class holding helper methods.

Definition at line 19 of file [Utilities.hpp](#).

25.32.2 Member Function Documentation

25.32.2.1 void RMOL::Utilities::computeDistributionParameters (const [UnconstrainedDemandVector_T](#) & *iVector*, double & *ioMean*, double & *ioStdDev*) [static]

Compute the mean and the standard deviation from a set of samples.

Definition at line 24 of file [Utilities.cpp](#).

25.32.2.2 stdair::DCPList_T RMOL::Utilities::buildRemainingDCPList (const stdair::DTD_T & *iDTD*) [static]

Build the list of remaining DCP's for the segment-date.

Definition at line 55 of file [Utilities.cpp](#).

References [RMOL::DEFAULT_DCP_LIST](#).

Referenced by [RMOL::Forecaster::forecastUsingAdditivePickUp\(\)](#).

25.32.2.3 stdair::DCPList_T RMOL::Utilities::buildRemainingDCPList2 (const stdair::DTD_T & *iDTD*) [static]

Definition at line 80 of file [Utilities.cpp](#).

References [RMOL::DEFAULT_DCP_LIST](#).

Referenced by [RMOL::Forecaster::forecastUsingAdditivePickUp\(\)](#).

25.32.2.4 `stdair::NbOfSegments_T RMOL::Utilities::getNbOfDepartedSimilarSegments (const stdair::SegmentCabin & iSegmentCabin, const stdair::Date_T & iEventDate) [static]`

Retrieve the number of departed similar segments.

Definition at line 105 of file [Utilities.cpp](#).

References [RMOL::GuillotineBlockHelper::getNbOfSegmentAlreadyPassedThisDTD\(\)](#).

The documentation for this class was generated from the following files:

- [rmol/bom/Utilities.hpp](#)
- [rmol/bom/Utilities.cpp](#)

26 File Documentation

26.1 [doc/local/authors.doc](#) File Reference

26.2 [doc/local/codingrules.doc](#) File Reference

26.3 [doc/local/copyright.doc](#) File Reference

26.4 [doc/local/documentation.doc](#) File Reference

26.5 [doc/local/features.doc](#) File Reference

26.6 [doc/local/help_wanted.doc](#) File Reference

26.7 [doc/local/howto_release.doc](#) File Reference

26.8 [doc/local/index.doc](#) File Reference

26.9 [doc/local/installation.doc](#) File Reference

26.10 [doc/local/linking.doc](#) File Reference

26.11 [doc/local/test.doc](#) File Reference

26.12 [doc/local/users_guide.doc](#) File Reference

26.13 [doc/local/verification.doc](#) File Reference

26.14 [doc/tutorial/tutorial.doc](#) File Reference

26.15 [rmol/basic/BasConst.cpp](#) File Reference

```
#include <rmol/basic/BasConst_General.hpp>
#include <rmol/basic/BasConst_Curves.hpp>
#include <rmol/basic/BasConst_RMOL_Service.hpp>
```

Namespaces

- namespace [RMOL](#)

Variables

- const stdair::AirlineCode_T [RMOL::DEFAULT_RMOL_SERVICE_AIRLINE_CODE](#) = "BA"
- const double [RMOL::DEFAULT_RMOL_SERVICE_CAPACITY](#) = 1.0
- const int [RMOL::DEFAULT_NUMBER_OF_DRAWS_FOR_MC_SIMULATION](#) = 100000
- const int [RMOL::DEFAULT_PRECISION](#) = 10
- const double [RMOL::DEFAULT_EPSILON](#) = 0.0001
- const double [RMOL::DEFAULT_STOPPING_CRITERION](#) = 0.01
- const double [RMOL::DEFAULT_INITIALIZER_DOUBLE_NEGATIVE](#) = -10.0
- const FRAT5Curve_T [RMOL::DEFAULT_CUMULATIVE_FRAT5_CURVE](#)
- const stdair::DCPList_T [RMOL::DEFAULT_DCP_LIST](#) = DefaultDCPList::init()

26.16 BasConst.cpp

```

00001 ///////////////////////////////////////////////////////////////////
00002 // Import section
00003 ///////////////////////////////////////////////////////////////////
00004 #include <rmol/basic/BasConst_General.hpp>
00005 #include <rmol/basic/BasConst_Curves.hpp>
00006 #include <rmol/basic/BasConst_RMOL_Service.hpp>
00007 >
00008 namespace RMOL {
00009
00011     const stdair::AirlineCode_T DEFAULT_RMOL_SERVICE_AIRLINE_CODE
00012     = "BA";
00014     const double DEFAULT_RMOL_SERVICE_CAPACITY = 1.0
00015 ;
00018     const int DEFAULT_NUMBER_OF_DRAWS_FOR_MC_SIMULATION
00019     = 100000;
00023     const int DEFAULT_PRECISION = 10;
00024
00026     const double DEFAULT_EPSILON = 0.0001;
00027
00029     const double DEFAULT_STOPPING_CRITERION = 0.01;
00030
00032     const double DEFAULT_INITIALIZER_DOUBLE_NEGATIVE
00033     = -10.0;
00036     const FRAT5Curve_T DEFAULT_CUMULATIVE_FRAT5_CURVE
00037     =
00038     DefaultMap::createCumulativeFRAT5Curve
00039     ();
00040     FRAT5Curve_T DefaultMap::createCumulativeFRAT5Curve
00041     () {
00042         FRAT5Curve_T oCurve;
00043         // oCurve[63] = 1.4; oCurve[56] = 1.45;
00044         // oCurve[49] = 1.5; oCurve[42] = 1.55; oCurve[35] = 1.6;
00045         // oCurve[31] = 1.7; oCurve[27] = 1.8; oCurve[23] = 2.0;
00046         // oCurve[19] = 2.3; oCurve[16] = 2.6; oCurve[13] = 3.0;
00047         // oCurve[10] = 3.3; oCurve[7] = 3.4; oCurve[5] = 3.44;
00048         // oCurve[3] = 3.47; oCurve[1] = 3.5;
00049         oCurve[63] = 1.1; oCurve[56] = 1.11;
00050         oCurve[49] = 1.17; oCurve[42] = 1.27;
00051         oCurve[35] = 1.28; oCurve[31] = 1.28; oCurve[27] = 1.28;
00052         oCurve[23] = 1.37; oCurve[19] = 1.37;
00053         oCurve[16] = 1.6; oCurve[13] = 1.6;
00054         oCurve[10] = 1.8; oCurve[7] = 1.8;
00055         oCurve[5] = 2.23; oCurve[3] = 2.23;
00056         oCurve[1] = 2.5;
00057         // oCurve[63] = 1.05; oCurve[56] = 1.07;
00058         // oCurve[49] = 1.09; oCurve[42] = 1.11; oCurve[35] = 1.14;
00059         // oCurve[31] = 1.16; oCurve[27] = 1.18; oCurve[23] = 1.21;
00060         // oCurve[19] = 1.24; oCurve[16] = 1.27; oCurve[13] = 1.3;
00061         // oCurve[10] = 1.33; oCurve[7] = 1.37; oCurve[5] = 1.4;
00062         // oCurve[3] = 1.45; oCurve[1] = 1.5;
00063         // oCurve[63] = 1.4;
00064         // oCurve[49] = 1.5; oCurve[35] = 1.6;
00065         // oCurve[23] = 2.0; oCurve[16] = 2.6;

```

```

00063     // oCurve[10] = 3.3;  oCurve[5]  = 3.44;
00064     // oCurve[1]  = 3.5;
00065     return oCurve;
00066 };
00067
00069 const stdair::DCPList_T DEFAULT_DCP_LIST =
DefaultDCPList::init();
00070 stdair::DCPList_T DefaultDCPList::init() {
00071     stdair::DCPList_T oDCPList;
00072     oDCPList.push_back (63); oDCPList.push_back (49);
00073     oDCPList.push_back (35); oDCPList.push_back (23);
00074     oDCPList.push_back (16); oDCPList.push_back (10);
00075     oDCPList.push_back (5);  oDCPList.push_back (1);
00076     oDCPList.push_back (0);
00077     return oDCPList;
00078 }
00079
00080 }

```

26.17 rmol/basic/BasConst_Curves.hpp File Reference

```
#include <rmol/RMOL_Types.hpp>
```

Classes

- struct [RMOL::DefaultMap](#)

Namespaces

- namespace [RMOL](#)

26.18 BasConst_Curves.hpp

```

00001 #ifndef __RMOL_BAS_BASCONST_CURVES_HPP
00002 #define __RMOL_BAS_BASCONST_CURVES_HPP
00003
00004 // //////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////
00007 // RMOL
00008 #include <rmol/RMOL_Types.hpp>
00009
00010 namespace RMOL {
00011
00014     extern const FRAT5Curve_T DEFAULT_CUMULATIVE_FRAT5_CURVE
00015 ;
00017     struct DefaultMap {
00018         static FRAT5Curve_T createCumulativeFRAT5Curve
00019         ();
00019     };
00020 }
00021 #endif // __RMOL_BAS_BASCONST_CURVES_HPP

```

26.19 rmol/basic/BasConst_General.hpp File Reference

```
#include <stdair/stdair_types.hpp>
```

Classes

- struct [RMOL::DefaultDCPList](#)

Namespaces

- namespace [RMOL](#)

26.20 BasConst_General.hpp

```

00001 #ifndef __RMOL_BAS_BASCONST_GENERAL_HPP
00002 #define __RMOL_BAS_BASCONST_GENERAL_HPP
00003
00004 // //////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////
00007 // StdAir
00008 #include <stdair/stdair_types.hpp>
00009
00010 namespace RMOL {
00011
00014     extern const int DEFAULT_NUMBER_OF_DRAWS_FOR_MC_SIMULATION
00015 ;
00018     extern const int DEFAULT_PRECISION;
00019
00021     extern const double DEFAULT_EPSILON;
00022
00024     extern const double DEFAULT_STOPPING_CRITERION;
00025
00027     extern const double DEFAULT_INITIALIZER_DOUBLE_NEGATIVE
00028 ;
00030     extern const stdair::DCPList_T DEFAULT_DCP_LIST;
00031     struct DefaultDCPList { static stdair::DCPList_T init(); };
00032 }
00033 #endif // __RMOL_BAS_BASCONST_GENERAL_HPP

```

26.21 rmol/basic/BasConst_RMOL_Service.hpp File Reference

```

#include <vector>
#include <stdair/stdair_basic_types.hpp>
#include <rmol/RMOL_Types.hpp>

```

Namespaces

- namespace [RMOL](#)

26.22 BasConst_RMOL_Service.hpp

```

00001 #ifndef __RMOL_BAS_BASCONST_RMOL_SERVICE_HPP
00002 #define __RMOL_BAS_BASCONST_RMOL_SERVICE_HPP
00003
00004 // //////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////
00007 // STL
00008 #include <vector>
00009 // StdAir
00010 #include <stdair/stdair_basic_types.hpp>
00011 // RMOL
00012 #include <rmol/RMOL_Types.hpp>
00013
00014 namespace RMOL {
00015
00017     extern const stdair::AirlineCode_T DEFAULT_RMOL_SERVICE_AIRLINE_CODE
00018 ;
00020     extern const double DEFAULT_RMOL_SERVICE_CAPACITY
00021 ;
00022 }
00023 #endif // __RMOL_BAS_BASCONST_RMOL_SERVICE_HPP

```

26.23 rmol/batches/rmol.cpp File Reference

```
#include <cassert>
#include <iostream>
#include <sstream>
#include <fstream>
#include <string>
#include <boost/date_time/posix_time/posix_time.hpp>
#include <boost/date_time/gregorian/gregorian.hpp>
#include <boost/program_options.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/RMOL_Service.hpp>
#include <rmol/config/rmol-paths.hpp>
```

Functions

- `const std::string K_RMOL_DEFAULT_LOG_FILENAME ("rmol.log")`
- `const std::string K_RMOL_DEFAULT_INPUT_FILENAME (STDAIR_SAMPLE_DIR"/rm01.csv")`
- `template<class T > std::ostream & operator<< (std::ostream &os, const std::vector< T > &v)`
- `int readConfiguration (int argc, char *argv[], int &ioRandomDraws, double &ioCapacity, short &ioMethod, bool &iolsBuiltin, std::string &iolInputFilename, std::string &iolLogFilename)`
- `void optimise (RMOL::RMOL_Service &rmolService, const short &iMethod, const int &iRandomDraws)`
- `int main (int argc, char *argv[])`

Variables

- `const bool K_RMOL_DEFAULT_BUILT_IN_INPUT = false`
- `const int K_RMOL_DEFAULT_RANDOM_DRAWS = 100000`
- `const double K_RMOL_DEFAULT_CAPACITY = 500.0`
- `const short K_RMOL_DEFAULT_METHOD = 0`
- `const int K_RMOL_EARLY_RETURN_STATUS = 99`

26.23.1 Function Documentation

26.23.1.1 `const std::string K_RMOL_DEFAULT_LOG_FILENAME ("rmol.log")`

Default name and location for the log file.

Referenced by [readConfiguration\(\)](#).

26.23.1.2 `const std::string K_RMOL_DEFAULT_INPUT_FILENAME (STDAIR_SAMPLE_DIR"/rm01.csv")`

Default name and location for the (CSV) input file.

Referenced by [readConfiguration\(\)](#).

26.23.1.3 `template<class T > std::ostream& operator<< (std::ostream & os, const std::vector< T > & v)`

Definition at line 47 of file [rmol.cpp](#).

26.23.1.4 `int readConfiguration (int argc, char * argv[], int & ioRandomDraws, double & ioCapacity, short & ioMethod, bool & iolsBuiltin, std::string & iolInputFilename, std::string & iolLogFilename)`

Read and parse the command line options.

Definition at line 57 of file [rmol.cpp](#).

References [K_RMOL_DEFAULT_BUILT_IN_INPUT](#), [K_RMOL_DEFAULT_CAPACITY](#), [K_RMOL_DEFAULT_INPUT_FILENAME\(\)](#), [K_RMOL_DEFAULT_LOG_FILENAME\(\)](#), [K_RMOL_DEFAULT_METHOD](#), [K_RMOL_DEFAULT_RANDOM_DRAWS](#), [K_RMOL_EARLY_RETURN_STATUS](#), [PACKAGE_NAME](#), [PACKAGE_VERSION](#), and [PREFIXDIR](#).

Referenced by [main\(\)](#).

26.23.1.5 `void optimise (RMOL::RMOL_Service & rmolService, const short & iMethod, const int & iRandomDraws)`

Definition at line 167 of file [rmol.cpp](#).

References [RMOL::RMOL_Service::heuristicOptimisationByEmsr\(\)](#), [RMOL::RMOL_Service::heuristicOptimisationByEmsrA\(\)](#), [RMOL::RMOL_Service::heuristicOptimisationByEmsrB\(\)](#), [RMOL::RMOL_Service::optimalOptimisationByDP\(\)](#), and [RMOL::RMOL_Service::optimalOptimisationByMCIntegration\(\)](#).

Referenced by [main\(\)](#).

26.23.1.6 `int main (int argc, char * argv[])`

Definition at line 204 of file [rmol.cpp](#).

References [RMOL::RMOL_Service::buildSampleBom\(\)](#), [K_RMOL_EARLY_RETURN_STATUS](#), [optimise\(\)](#), [RMOL::RMOL_Service::parseAndLoad\(\)](#), and [readConfiguration\(\)](#).

26.23.2 Variable Documentation

26.23.2.1 `const bool K_RMOL_DEFAULT_BUILT_IN_INPUT = false`

Default for the input type. It can be either built-in or provided by an input file. That latter must then be given with the -i/-input option.

Definition at line 23 of file [rmol.cpp](#).

Referenced by [readConfiguration\(\)](#).

26.23.2.2 `const int K_RMOL_DEFAULT_RANDOM_DRAWS = 100000`

Default number of random draws to be generated (best if over 100).

Definition at line 29 of file [rmol.cpp](#).

Referenced by [readConfiguration\(\)](#).

26.23.2.3 `const double K_RMOL_DEFAULT_CAPACITY = 500.0`

Default value for the capacity of the resource (e.g., a flight cabin).

Definition at line 32 of file [rmol.cpp](#).

Referenced by [readConfiguration\(\)](#).

26.23.2.4 `const short K_RMOL_DEFAULT_METHOD = 0`

Default name and location for the Revenue Management method to be used.

- 0 = Monte-Carlo
- 1 = Dynamic Programming
- 2 = EMSR
- 3 = EMSR-a
- 4 = EMSR-b

Definition at line 43 of file [rmol.cpp](#).

Referenced by [readConfiguration\(\)](#).

26.23.2.5 const int K_RMOL_EARLY_RETURN_STATUS = 99

Early return status (so that it can be differentiated from an error).

Definition at line 54 of file [rmol.cpp](#).

Referenced by [main\(\)](#), and [readConfiguration\(\)](#).

26.24 rmol.cpp

```

00001 // STL
00002 #include <cassert>
00003 #include <iostream>
00004 #include <sstream>
00005 #include <fstream>
00006 #include <string>
00007 // Boost (Extended STL)
00008 #include <boost/date_time/posix_time/posix_time.hpp>
00009 #include <boost/date_time/gregorian/gregorian.hpp>
00010 #include <boost/program_options.hpp>
00011 // StdAir
00012 #include <stdair/service/Logger.hpp>
00013 // RMOL
00014 #include <rmol/RMOL_Service.hpp>
00015 #include <rmol/config/rmol-paths.hpp>
00016
00017 // ////////// Constants //////////
00019 const std::string K_RMOL_DEFAULT_LOG_FILENAME ("
    rmol.log");
00020
00023 const bool K_RMOL_DEFAULT_BUILT_IN_INPUT = false;
00024
00026 const std::string K_RMOL_DEFAULT_INPUT_FILENAME (
    STDAIR_SAMPLE_DIR "/rmol.csv");
00027
00029 const int K_RMOL_DEFAULT_RANDOM_DRAWS = 100000;
00030
00032 const double K_RMOL_DEFAULT_CAPACITY = 500.0;
00033
00043 const short K_RMOL_DEFAULT_METHOD = 0;
00044
00045 // ////////// Parsing of Options & Configuration //////////
00046 // A helper function to simplify the main part.
00047 template<class T> std::ostream& operator<< (std::ostream& os,
00048     const std::vector<T>& v) {
00049     std::copy (v.begin(), v.end(), std::ostream_iterator<T> (std::cout, " "));
00050     return os;
00051 }
00052
00054 const int K_RMOL_EARLY_RETURN_STATUS = 99;
00055
00057 int readConfiguration(int argc, char* argv[],
00058     int& ioRandomDraws, double& ioCapacity,
00059     short& ioMethod, bool& ioIsBuiltin,
00060     std::string& ioInputFilename, std::string& ioLogFilename)
00061 {
00062     // Default for the built-in input
00063     ioIsBuiltin = K_RMOL_DEFAULT_BUILT_IN_INPUT;
00064
00065     // Declare a group of options that will be allowed only on command line
00066     boost::program_options::options_description generic ("Generic options");
00067     generic.add_options()
00068         ("prefix", "print installation prefix")
00069         ("version,v", "print version string")
00070         ("help,h", "produce help message");
00071
00072     // Declare a group of options that will be allowed both on command
00073     // line and in config file
00074     boost::program_options::options_description config ("Configuration");
00075     config.add_options()
00076         ("draws,d",
00077         boost::program_options::value<int>(&ioRandomDraws)->default_value(
00078             K_RMOL_DEFAULT_RANDOM_DRAWS),
00079         "Number of to-be-generated random draws")
00079         ("capacity,c",
00080         boost::program_options::value<double>(&ioCapacity)->default_value(
00081             K_RMOL_DEFAULT_CAPACITY),

```

```

00081     "Resource capacity (e.g., for a flight leg)")
00082     ("method,m",
00083     boost::program_options::value<short>(&ioMethod)->default_value(
K_RMOL_DEFAULT_METHOD),
00084     "Revenue Management method to be used (0 = Monte-Carlo, 1 = Dynamic
Programming, 2 = EMSR, 3 = EMSR-a, 4 = EMSR-b)")
00085     ("builtin,b",
00086     "The cabin set up can be either built-in or parsed from an input file.
That latter must then be given with the -i/--input option")
00087     ("input,i",
00088     boost::program_options::value< std::string >(&ioInputFilename)->
default_value(K_RMOL_DEFAULT_INPUT_FILENAME),
00089     "(CSV) input file for the demand distribution parameters and resource
(leg-cabin) capacities")
00090     ("log,l",
00091     boost::program_options::value< std::string >(&ioLogFilename)->
default_value(K_RMOL_DEFAULT_LOG_FILENAME),
00092     "Filename for the logs")
00093     ;
00094
00095     // Hidden options, will be allowed both on command line and
00096     // in config file, but will not be shown to the user.
00097     boost::program_options::options_description hidden ("Hidden options");
00098     hidden.add_options()
00099     ("copyright",
00100     boost::program_options::value< std::vector<std::string> >(),
00101     "Show the copyright (license)");
00102
00103     boost::program_options::options_description cmdline_options;
00104     cmdline_options.add(generic).add(config).add(hidden);
00105
00106     boost::program_options::options_description config_file_options;
00107     config_file_options.add(config).add(hidden);
00108
00109     boost::program_options::options_description visible ("Allowed options");
00110     visible.add(generic).add(config);
00111
00112     boost::program_options::positional_options_description p;
00113     p.add ("copyright", -1);
00114
00115     boost::program_options::variables_map vm;
00116     boost::program_options::
00117     store (boost::program_options::command_line_parser (argc, argv).
00118     options (cmdline_options).positional(p).run(), vm);
00119
00120     std::ifstream ifs ("rmol.cfg");
00121     boost::program_options::store (parse_config_file (ifs, config_file_options),
00122     vm);
00123     boost::program_options::notify (vm);
00124
00125     if (vm.count ("help")) {
00126         std::cout << visible << std::endl;
00127         return K_RMOL_EARLY_RETURN_STATUS;
00128     }
00129
00130     if (vm.count ("version")) {
00131         std::cout << PACKAGE_NAME << ", version " << PACKAGE_VERSION
<< std::endl;
00132         return K_RMOL_EARLY_RETURN_STATUS;
00133     }
00134
00135     if (vm.count ("prefix")) {
00136         std::cout << "Installation prefix: " << PREFIXDIR << std::endl;
00137         return K_RMOL_EARLY_RETURN_STATUS;
00138     }
00139
00140     if (vm.count ("builtin")) {
00141         ioIsBuiltin = true;
00142     }
00143     const std::string isBuiltinStr = (ioIsBuiltin == true)?"yes":"no";
00144     std::cout << "The BOM should be built-in? " << isBuiltinStr << std::endl;
00145
00146     if (ioIsBuiltin == false) {
00147         if (vm.count ("input")) {
00148             ioInputFilename = vm["input"].as< std::string >();
00149             std::cout << "Input filename is: " << ioInputFilename << std::endl;
00150         }
00151     }
00152
00153     if (vm.count ("log")) {
00154         ioLogFilename = vm["log"].as< std::string >();
00155         std::cout << "Log filename is: " << ioLogFilename << std::endl;
00156     }
00157
00158     std::cout << "The number of random draws is: " << ioRandomDraws << std::endl;
00159     std::cout << "The resource capacity is: " << ioCapacity << std::endl;
00160     std::cout << "The optimisation method is: " << ioMethod << std::endl;

```

```

00161     std::cout << std::endl;
00162
00163     return 0;
00164 }
00165
00166 // //////////////////////////////////////
00167 void optimise (RMOL::RMOL_Service& rmolService,
00168               const short& iMethod, const int& iRandomDraws) {
00169
00170     switch (iMethod) {
00171     case 0: {
00172         // Calculate the optimal protections by the Monte Carlo
00173         // Integration approach
00174         rmolService.optimalOptimisationByMCIntegration
00175         (iRandomDraws);
00176         break;
00177     }
00178     case 1: {
00179         // Calculate the optimal protections by DP.
00180         rmolService.optimalOptimisationByDP ();
00181         break;
00182     }
00183     case 2: {
00184         // Calculate the Bid-Price Vector by EMSR
00185         rmolService.heuristicOptimisationByEmsr ();
00186         break;
00187     }
00188     case 3: {
00189         // Calculate the protections by EMSR-a
00190         rmolService.heuristicOptimisationByEmsrA ();
00191         break;
00192     }
00193     case 4: {
00194         // Calculate the protections by EMSR-b
00195         rmolService.heuristicOptimisationByEmsrB ();
00196         break;
00197     }
00198     default: {
00199         rmolService.optimalOptimisationByMCIntegration
00200         (iRandomDraws);
00201     }
00202 }
00203 // ////////////////////////////////// M A I N //////////////////////////////////
00204 int main (int argc, char* argv[]) {
00205
00206     // Number of random draws to be generated (best if greater than 100)
00207     int lRandomDraws = 0;
00208
00209     // Cabin Capacity (it must be greater then 100 here)
00210     double lCapacity = 0.0;
00211
00212     // Methods of optimisation (0 = Monte-Carlo, 1 = Dynamic Programming,
00213     // 2 = EMSR, 3 = EMSR-a, 4 = EMSR-b)
00214     short lMethod = 0;
00215
00216     // Built-in
00217     bool isBuiltin;
00218
00219     // Input file name
00220     std::string lInputFilename;
00221
00222     // Output log File
00223     std::string lLogFilename;
00224
00225     // Call the command-line option parser
00226     const int lOptionParserStatus =
00227     readConfiguration (argc, argv, lRandomDraws, lCapacity,
00228     lMethod,
00229     isBuiltin, lInputFilename, lLogFilename);
00230
00231     if (lOptionParserStatus == K_RMOL_EARLY_RETURN_STATUS
00232     ) {
00233         return 0;
00234     }
00235
00236     // Set the log parameters
00237     std::ofstream logOutputFile;
00238     // Open and clean the log outputfile
00239     logOutputFile.open (lLogFilename.c_str());
00240     logOutputFile.clear();
00241
00242     // Initialise the log stream
00243     const stdair::BasLogParams lLogParams (stdair::LOG::DEBUG, logOutputFile);
00244
00245     // Initialise the RMOL service

```

```

00244   RMOL::RMOL_Service rmolService (lLogParams);
00245
00246   if (isBuiltin == true) {
00247       // DEBUG
00248       STDAIR_LOG_DEBUG ("No input file has been given."
00249           "A sample BOM tree will therefore be built.");
00250
00251       // Build a sample BOM tree
00252       rmolService.buildSampleBom();
00253   } else {
00254       // DEBUG
00255       STDAIR_LOG_DEBUG ("RMOL will parse " << lInputFilename
00256           << " and build the corresponding BOM tree.");
00257
00258       //
00259       rmolService.parseAndLoad (lCapacity, lInputFilename);
00260   }
00261
00262   // Launch the optimisation
00263   optimise (rmolService, lMethod, lRandomDraws);
00264
00265   //
00266   logOutputFile.close();
00267
00268   return 0;
00269 }
00270

```

26.25 rmol/bom/BucketHolderTypes.hpp File Reference

```

#include <list>
#include <map>
#include <stdair/stdair_basic_types.hpp>

```

Namespaces

- namespace [RMOL](#)

Typedefs

- typedef std::list< BucketHolder * > [RMOL::BucketHolderList_T](#)

26.26 BucketHolderTypes.hpp

```

00001 #ifndef __RMOL_BUCKETHOLDERTYPES_HPP
00002 #define __RMOL_BUCKETHOLDERTYPES_HPP
00003
00004 // //////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////
00007 // STL
00008 #include <list>
00009 #include <map>
00010 // STDAIR
00011 #include <stdair/stdair_basic_types.hpp>
00012
00013 namespace RMOL {
00014
00015     class BucketHolder;
00016
00017     typedef std::list<BucketHolder*> BucketHolderList_T;
00018
00019     typedef std::map<const stdair::MapKey_T, BucketHolder*>;
00020 }
00021
00022 #endif // __RMOL_BUCKETHOLDERTYPES_HPP

```

26.27 rmol/bom/DistributionParameterList.hpp File Reference

```

#include <list>

```

```
#include <rmol/field/FldDistributionParameters.hpp>
```

Namespaces

- namespace [RMOL](#)

Typedefs

- typedef std::list
< FldDistributionParameters > [RMOL::DistributionParameterList_T](#)

26.28 DistributionParameterList.hpp

```
00001 #ifndef __RMOL_DISTRIBUTIONPARAMETERLIST_HPP
00002 #define __RMOL_DISTRIBUTIONPARAMETERLIST_HPP
00003
00004 // //////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////
00007 // STL
00008 #include <list>
00009 // RMOL
00010 #include <rmol/field/FldDistributionParameters.hpp>
00011
00012 namespace RMOL {
00013
00016     typedef std::list<FldDistributionParameters> DistributionParameterList_T
    ;
00017
00018 }
00019 #endif // __RMOL_DISTRIBUTIONPARAMETERLIST_HPP
```

26.29 rmol/bom/DPOptimiser.cpp File Reference

```
#include <cassert>
#include <sstream>
#include <vector>
#include <cmath>
#include <boost/math/distributions/normal.hpp>
#include <stdair/bom/LegCabin.hpp>
#include <stdair/bom/VirtualClassStruct.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/basic/BasConst_General.hpp>
#include <rmol/bom/DPOptimiser.hpp>
```

Namespaces

- namespace [RMOL](#)

26.30 DPOptimiser.cpp

```
00001 // //////////////////////////////////////
00002 // Import section
00003 // //////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 #include <sstream>
00007 #include <vector>
00008 #include <cmath>
00009 // Boost Math
00010 #include <boost/math/distributions/normal.hpp>
```

```

00011 // StdAir
00012 #include <stdair/bom/LegCabin.hpp>
00013 #include <stdair/bom/VirtualClassStruct.hpp>
00014 #include <stdair/service/Logger.hpp>
00015 // RMOL
00016 #include <rmol/basic/BasConst_General.hpp>
00017 #include <rmol/bom/DPOptimiser.hpp>
00018
00019 namespace RMOL {
00020
00021 ///////////////////////////////////////////////////////////////////
00022 void DPOptimiser::optimalOptimisationByDP
00023 (stdair::LegCabin& ioLegCabin) {
00024     // // Number of classes/buckets: n
00025     // const short nbOfClasses = ioBucketHolder.getSize();
00026     // // Number of values of x to compute for each Vj(x).
00027     // const int maxValue = static_cast<int> (iCabinCapacity *
00028     DEFAULT_PRECISION);
00029     // // Vector of the Expected Maximal Revenue (Vj).
00030     // std::vector< std::vector<double> > MERVectorHolder;
00031     // // Vector of V_0(x).
00032     // std::vector<double> initialMERVector (maxValue+1, 0.0);
00033     // MERVectorHolder.push_back (initialMERVector);
00034     // // Current cumulative protection level (y_j * DEFAULT_PRECISION).
00035     // // Initialise with y_0 = 0.
00036     // int currentProtection = 0;
00037     // int currentBucketIndex = 1;
00038     // ioBucketHolder.begin();
00039     // while (currentProtection < maxValue && currentBucketIndex < nbOfClasses)
00040     {
00041         // //while (currentBucketIndex == 1) {
00042         // bool protectionChanged = false;
00043         // double nextProtection = 0.0;
00044         // std::vector<double> currentMERVector;
00045         // // double testGradient = 10000;
00046         //
00047         // Bucket& currentBucket = ioBucketHolder.getCurrentBucket();
00048         // const double meanDemand = currentBucket.getMean();
00049         // const double SDDemand = currentBucket.getStandardDeviation();
00050         // const double currentYield = currentBucket.getAverageYield();
00051         // const double errorFactor = 1.0;
00052         //
00053         // Bucket& nextBucket = ioBucketHolder.getNextBucket();
00054         // const double nextYield = nextBucket.getAverageYield();
00055         //
00056         // // For x <= currentProtection (y_(j-1)), V_j(x) = V_(j-1)(x).
00057         // for (int x = 0; x <= currentProtection; ++x) {
00058         //     const double MERValue =
00059         MERVectorHolder.at(currentBucketIndex-1).at(x);
00060         //     currentMERVector.push_back (MERValue);
00061         // }
00062         // //
00063         // boost::math::normal lNormalDistribution (meanDemand, SDDemand);
00064         //
00065         // // Vector of gaussian pdf values.
00066         // std::vector<double> pdfVector;
00067         // for (int s = 0; s <= maxValue - currentProtection; ++s) {
00068         //     const double pdfValue =
00069         //         boost::math::pdf (lNormalDistribution, s/DEFAULT_PRECISION);
00070         //     pdfVector.push_back (pdfValue);
00071         // }
00072         //
00073         // // Vector of gaussian cdf values.
00074         // std::vector<double> cdfVector;
00075         // for (int s = 0; s <= maxValue - currentProtection; ++s) {
00076         //     const double cdfValue =
00077         //         boost::math::cdf (boost::math::complement (lNormalDistribution,
00078         //                                                     s/DEFAULT_PRECISION));
00079         //     cdfVector.push_back (cdfValue);
00080         // }
00081         //
00082         // // Compute V_j(x) for x > currentProtection (y_(j-1)).
00083         // for (int x = currentProtection + 1; x <= maxValue; ++x) {
00084         //     const double lowerBound = static_cast<double> (x -
00085         currentProtection);
00086         //
00087         // // Compute the first integral in the V_j(x) formulation (see
00088         // // the memo of Jerome Contant).
00089         // const double power1 =
00090         //     - 0.5 * meanDemand * meanDemand / (SDDemand * SDDemand);

```

```

00093 //      const double e1 = std::exp (power1);
00094 //      const double power2 =
00095 //          - 0.5 * (lowerBound / DEFAULT_PRECISION - meanDemand)
00096 //          * (lowerBound / DEFAULT_PRECISION - meanDemand)
00097 //          / (SDDemand * SDDemand);
00098 //      const double e2 = std::exp (power2);
00099
00100 //      const double cdfValue0 =
00101 //          boost::math::cdf (boost::math::complement (lNormalDistribution,
00102 //              0.0));
00103 //      const double cdfValue1 =
00104 //          boost::math::cdf(boost::math::complement(lNormalDistribution,
00105 //              lowerBound/DEFAULT_PRECISION));
00106 //      const double integralResult1 = currentYield
00107 //          * ((e1 - e2) * SDDemand / sqrt (2 * 3.14159265)
00108 //          + meanDemand * (cdfValue0 - cdfValue1));
00109
00110 //      double integralResult2 = 0.0;
00111
00112 //      for (int s = 0; s < lowerBound; ++s) {
00113 //          const double partialResult =
00114 //              2 * MERVectorHolder.at (currentBucketIndex-1).at (x-s)
00115 //              * pdfVector.at (s);
00116 //          integralResult2 += partialResult;
00117 //      }
00118 //      integralResult2 -= MERVectorHolder.at (currentBucketIndex-1).at (x) *
00119 //          pdfVector.at (0);
00120
00121 //      const int intLowerBound = static_cast<int>(lowerBound);
00122 //      integralResult2 +=
00123 //          MERVectorHolder.at (currentBucketIndex-1).at (x - intLowerBound) *
00124 //          pdfVector.at (intLowerBound);
00125
00126 //      integralResult2 /= 2 * DEFAULT_PRECISION;
00127 //      /*
00128 //      for (int s = 0; s < lowerBound; ++s) {
00129 //          const double partialResult =
00130 //              (MERVectorHolder.at (currentBucketIndex-1).at (x-s) +
00131 //              MERVectorHolder.at (currentBucketIndex-1).at (x-s-1)) *
00132 //              (cdfVector.at (s+1) - cdfVector.at (s)) / 2;
00133 //          integralResult2 += partialResult;
00134 //      }
00135 //      */
00136 //      const double firstElement = integralResult1 + integralResult2;
00137
00138 //      // Compute the second integral in the V_j(x) formulation (see
00139 //      // the memo of Jerome Contant).
00140 //      const double constCoefOfSecondElement =
00141 //          currentYield * lowerBound / DEFAULT_PRECISION
00142 //          + MERVectorHolder.at (currentBucketIndex-1).at (currentProtection);
00143
00144 //      const double secondElement = constCoefOfSecondElement
00145 //          * boost::math::cdf (boost::math::complement (lNormalDistribution,
00146 //              lowerBound/DEFAULT_PRECISION));
00147
00148 //      const double MERValue = (firstElement + secondElement) /
00149 //          errorFactor;
00150
00151 //      assert (currentMERVector.size() > 0);
00152 //      const double lastMERValue = currentMERVector.back();
00153
00154 //      const double currentGradient =
00155 //          (MERValue - lastMERValue) * DEFAULT_PRECISION;
00156
00157 //      //assert (currentGradient >= 0);
00158 //      if (currentGradient < -0) {
00159 //          std::ostringstream ostr;
00160 //          ostr << currentGradient << std::endl
00161 //              << "x = " << x << std::endl
00162 //              << "class: " << currentBucketIndex << std::endl;
00163 //          STDAIR_LOG_DEBUG (ostr.str());
00164 //      }
00165 //      /*
00166 //      assert (currentGradient <= testGradient);
00167 //      testGradient = currentGradient;
00168 //      */
00169 //      if (protectionChanged == false && currentGradient <= nextYield) {
00170 //          nextProtection = x - 1;
00171 //          protectionChanged = true;
00172 //      }
00173 //      if (protectionChanged == true && currentGradient > nextYield) {

```

```

00177     //      protectionChanged = false;
00178     //      }
00179
00180     //      if (protectionChanged == false && x == maxValue) {
00181     //          nextProtection = maxValue;
00182     //      }
00183
00184     //      currentMERVector.push_back (MERValue);
00185     //  }
00186
00187     //  // DEBUG
00188     //  STDAIR_LOG_DEBUG ("Vmaxindex = " << currentMERVector.back());
00189
00190     //  MERVectorHolder.push_back (currentMERVector);
00191
00192     //  const double realProtection = nextProtection / DEFAULT_PRECISION;
00193     //  const double bookingLimit = iCabinCapacity - realProtection;
00194
00195     //  currentBucket.setCumulatedProtection (realProtection);
00196     //  nextBucket.setCumulatedBookingLimit (bookingLimit);
00197
00198     //  currentProtection = static_cast<int> (std::floor (nextProtection));
00199
00200     //  ioBucketHolder.iterate();
00201     //  ++currentBucketIndex;
00202     //  }
00203 }
00204
00205 }
```

26.31 rmol/bom/DPOptimiser.hpp File Reference

```
#include <rmol/RMOL_Types.hpp>
```

Classes

- class [RMOL::DPOptimiser](#)

Namespaces

- namespace [stdair](#)
- namespace [RMOL](#)

26.32 DPOptimiser.hpp

```

00001 #ifndef __RMOL_BOM_DPOPTIMISER_HPP
00002 #define __RMOL_BOM_DPOPTIMISER_HPP
00003
00004 // //////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////
00007 // RMOL
00008 #include <rmol/RMOL_Types.hpp>
00009
00011 namespace stdair {
00012     class LegCabin;
00013 }
00014
00015 namespace RMOL {
00017     class DPOptimiser {
00018     public:
00019
00025         static void optimalOptimisationByDP (
00026             stdair::LegCabin&);
00026
00030         static double cdfGaussianQ (const double, const double);
00031     };
00032 }
00033 #endif // __RMOL_BOM_DPOPTIMISER_HPP
```


26.33 rmol/bom/EMDetruncator.cpp File Reference

```
#include <iostream>
#include <cmath>
#include <vector>
#include <cassert>
#include <stdair/stdair_basic_types.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/bom/HistoricalBookingHolder.hpp>
#include <rmol/bom/EMDetruncator.hpp>
```

Namespaces

- namespace [RMOL](#)

26.34 EMDetruncator.cpp

```
00001 // //////////////////////////////////////
00002 // Import section
00003 // //////////////////////////////////////
00004 // STL
00005 #include <iostream>
00006 #include <cmath>
00007 #include <vector>
00008 #include <cassert>
00009 // StdAir
00010 #include <stdair/stdair_basic_types.hpp>
00011 #include <stdair/service/Logger.hpp>
00012 // RMOL
00013 #include <rmol/bom/HistoricalBookingHolder.hpp>
00014 #include <rmol/bom/EMDetruncator.hpp>
00015
00016 namespace RMOL {
00017
00018 // //////////////////////////////////////
00019 void EMDetruncator::unconstrainUsingEMMethod
00020 (HistoricalBookingHolder& ioHistoricalBookingHolder) {
00021
00022     // Number of flights.
00023     const short lNbOfFlights =
00024         ioHistoricalBookingHolder.getNbOfFlights();
00025
00026     // Number of uncensored booking data.
00027     const short lNbOfUncensoredData =
00028         ioHistoricalBookingHolder.getNbOfUncensoredData();
00029
00030     if (lNbOfUncensoredData > 1) {
00031         // Number of uncensored bookings.
00032         const stdair::NbOfBookings_T lNbOfUncensoredBookings =
00033             ioHistoricalBookingHolder.getNbOfUncensoredBookings
00034     );
00035
00036     const double lMeanOfUncensoredBookings =
00037         static_cast<double>(lNbOfUncensoredBookings/lNbOfUncensoredData);
00038
00039     const double lStdDevOfUncensoredBookings =
00040         ioHistoricalBookingHolder.getUncensoredStandardDeviation
00041         (lMeanOfUncensoredBookings, lNbOfUncensoredData);
00042
00043     std::vector<bool> toBeUnconstrained =
00044         ioHistoricalBookingHolder.getListOfToBeUnconstrainedFlags
00045     );
00046
00047     double lDemandMean = lMeanOfUncensoredBookings;
00048     double lStdDev = lStdDevOfUncensoredBookings;
00049
00050     // DEBUG
00051     STDAIR_LOG_DEBUG ("mean: " << lDemandMean << ", std: " << lStdDev);
00052
00053     if (lStdDev != 0) {
00054         bool stopUnconstraining = false;
00055         while (stopUnconstraining == false) {
00056             stopUnconstraining = true;
```

```

00056         for (short i = 0; i < lNbOfFlights; ++i) {
00057             if (toBeUnconstrained.at(i) == true) {
00058                 // Get the unconstrained demand of the (i+1)-th flight.
00059                 const stdair::NbOfBookings_T demand =
00060                     ioHistoricalBookingHolder.getUnconstrainedDemand
00061             (i);
00062                 //STDAIR_LOG_DEBUG ("demand: " << demand);
00063                 // Execute the Expectation step.
00064                 const stdair::NbOfBookings_T expectedDemand =
00065                     ioHistoricalBookingHolder.
00066                         calculateExpectedDemand (lDemandMean, lStdDev, i, demand);
00067                 //STDAIR_LOG_DEBUG ("expected: " << expectedDemand);
00068                 assert (expectedDemand >= 0 || expectedDemand < 0);
00069
00070                 double absDiff =
00071                     static_cast<double>(expectedDemand - demand);
00072
00073                 if (absDiff < 0) {
00074                     absDiff = - absDiff;
00075                 }
00076                 if (absDiff < 0.001) {
00077                     toBeUnconstrained.at (i) = false;
00078                 }
00079                 else {
00080                     stopUnconstraining = false;
00081                 }
00082
00083                 ioHistoricalBookingHolder.setUnconstrainedDemand
00084             (expectedDemand,
00085                                     i);
00086             }
00087         }
00088         if (stopUnconstraining == false) {
00089             lDemandMean = ioHistoricalBookingHolder.getDemandMean(
00090 );
00091             lStdDev =
00092                 ioHistoricalBookingHolder.getStandardDeviation
00093             (lDemandMean);
00094         }
00095     }
00096 }
00097 }
00098 }

```

26.35 rmol/bom/EMDetruncator.hpp File Reference

Classes

- class [RMOL::EMDetruncator](#)

Namespaces

- namespace [RMOL](#)

26.36 EMDetruncator.hpp

```

00001 #ifndef __RMOL_BOM_EMDETUNCATOR_HPP
00002 #define __RMOL_BOM_EMDETUNCATOR_HPP
00003
00004 // //////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////
00007 namespace RMOL {
00008     // Forward declarations.
00009     struct HistoricalBookingHolder;
00010
00012     class EMDetruncator {
00013     public:
00016         static void unconstrainUsingEMMethod (
00017             HistoricalBookingHolder&);
00018     };
00019 #endif // __RMOL_BOM_EMDETUNCATOR_HPP

```

26.37 rmol/bom/Emsr.cpp File Reference

```

#include <assert.h>
#include <iostream>
#include <cmath>
#include <list>
#include <algorithm>
#include <stdair/stdair_rm_types.hpp>
#include <stdair/bom/LegCabin.hpp>
#include <stdair/bom/VirtualClassStruct.hpp>
#include <rmol/bom/Emsr.hpp>
#include <rmol/bom/EmsrUtils.hpp>

```

Namespaces

- namespace [RMOL](#)

26.38 Emsr.cpp

```

00001 // //////////////////////////////////////
00002 // Import section
00003 // //////////////////////////////////////
00004 // C
00005 #include <assert.h>
00006 // STL
00007 #include <iostream>
00008 #include <cmath>
00009 #include <list>
00010 #include <algorithm>
00011 // StdAir
00012 #include <stdair/stdair_rm_types.hpp>
00013 #include <stdair/bom/LegCabin.hpp>
00014 #include <stdair/bom/VirtualClassStruct.hpp>
00015 // RMOL
00016 #include <rmol/bom/Emsr.hpp>
00017 #include <rmol/bom/EmsrUtils.hpp>
00018
00019 namespace RMOL {
00020 // //////////////////////////////////////
00021 void Emsr::heuristicOptimisationByEmsrA (
    stdair::LegCabin& ioLegCabin) {
00022     stdair::VirtualClassList_T& lVirtualClassList =
00023         ioLegCabin.getVirtualClassList ();
00024     const stdair::CabinCapacity_T& lCabinCapacity =
00025         ioLegCabin.getOfferedCapacity();
00026
00032     stdair::VirtualClassList_T::iterator itVC = lVirtualClassList.begin();
00033     assert (itVC != lVirtualClassList.end());
00034
00035     stdair::VirtualClassStruct& lFirstVC = *itVC;
00036     lFirstVC.setCumulatedBookingLimit (lCabinCapacity);
00037     ++itVC;
00038     for (; itVC != lVirtualClassList.end(); ++itVC) {
00039         stdair::VirtualClassStruct& lNextVC = *itVC;
00040
00041         // Initialise the protection for class/bucket j.
00042         stdair::ProtectionLevel_T lProtectionLevel = 0.0;
00043
00044         for(stdair::VirtualClassList_T::iterator itHigherVC =
00045             lVirtualClassList.begin(); itHigherVC != itVC; ++itHigherVC) {
00046             stdair::VirtualClassStruct& lHigherVC = *itHigherVC;
00047             const double lPartialProtectionLevel =
00048                 EmsrUtils::computeProtectionLevel (
00049                     lHigherVC, lNextVC);
00049             lProtectionLevel += lPartialProtectionLevel;
00050         }
00051         stdair::VirtualClassList_T::iterator itCurrentVC = itVC; --itCurrentVC;
00052         stdair::VirtualClassStruct& lCurrentVC = *itCurrentVC;
00053         lCurrentVC.setCumulatedProtection (lProtectionLevel);
00054
00055         // Compute the booking limit for the class/bucket j+1 (can be negative).
00056         const double lBookingLimit = lCabinCapacity - lProtectionLevel;
00057
00058         // Set the booking limit for class/bucket j+1.

```

```

00059         lNextVC.setCumulatedBookingLimit (lBookingLimit);
00060     }
00061 }
00062
00063 // //////////////////////////////////////
00064 void Emsr::heuristicOptimisationByEmsrB (
stdair::LegCabin& ioLegCabin) {
00065     stdair::VirtualClassList_T& lVirtualClassList =
00066         ioLegCabin.getVirtualClassList ();
00067     const stdair::CabinCapacity_T& lCabinCapacity =
00068         ioLegCabin.getOfferedCapacity();
00069
00070     stdair::VirtualClassList_T::iterator itVC = lVirtualClassList.begin();
00071     assert (itVC != lVirtualClassList.end());
00072
00073     stdair::VirtualClassStruct& lFirstVC = *itVC;
00074     lFirstVC.setCumulatedBookingLimit (lCabinCapacity);
00075     ++itVC;
00076     stdair::VirtualClassStruct lAggregatedVC = lFirstVC;
00077     for (; itVC != lVirtualClassList.end(); ++itVC) {
00078         stdair::VirtualClassStruct& lNextVC = *itVC;
00079
00080         // Compute the protection level for the aggregated class/bucket
00081         // using the Little-Wood formular.
00082         const stdair::ProtectionLevel_T lProtectionLevel =
00083             EmsrUtils::computeProtectionLevel (
lAggregatedVC, lNextVC);
00084
00085         // Set the protection level for class/bucket j.
00086         stdair::VirtualClassList_T::iterator itCurrentVC = itVC; --itCurrentVC;
00087         stdair::VirtualClassStruct& lCurrentVC = *itCurrentVC;
00088         lCurrentVC.setCumulatedProtection (lProtectionLevel);
00089
00090         // Compute the booking limit for class/bucket j+1 (can be negative).
00091         const double lBookingLimit = lCabinCapacity - lProtectionLevel;
00092
00093         // Set the booking limit for class/bucket j+1.
00094         lNextVC.setCumulatedBookingLimit (lBookingLimit);
00095
00096         // Compute the aggregated class/bucket of classes/buckets 1,...,j.
00097         EmsrUtils::computeAggregatedVirtualClass
(lAggregatedVC, lNextVC);
00098     }
00099 }
00100
00101 // //////////////////////////////////////
00102 void Emsr::heuristicOptimisationByEmsr (
stdair::LegCabin& ioLegCabin) {
00103     stdair::VirtualClassList_T& lVirtualClassList =
00104         ioLegCabin.getVirtualClassList ();
00105     const stdair::CabinCapacity_T& lCapacity = ioLegCabin.getOfferedCapacity();
00106     ioLegCabin.emptyBidPriceVector();
00107     stdair::BidPriceVector_T& lBidPriceVector =
00108         ioLegCabin.getBidPriceVector();
00109
00110     // Cabin capacity in integer.
00111     const int lCabinCapacity = static_cast<const int> (lCapacity);
00112
00113     // List of all EMSR values.
00114     stdair::EmsrValueList_T lEmsrValueList;
00115
00116     for (stdair::VirtualClassList_T::iterator itVC = lVirtualClassList.begin();
itVC != lVirtualClassList.end(); ++itVC) {
00117         stdair::VirtualClassStruct& lCurrentVC = *itVC;
00118         for (int k = 1; k <= lCabinCapacity; ++k) {
00119             const double emsrValue = EmsrUtils::computeEmsrValue
(k, lCurrentVC);
00120             lEmsrValueList.push_back (emsrValue);
00121         }
00122     }
00123
00124     // Sort the EMSR values from high to low.
00125     std::sort (lEmsrValueList.rbegin(), lEmsrValueList.rend());
00126
00127     // Sanity check
00128     const int lEmsrValueListSize = lEmsrValueList.size();
00129     assert (lEmsrValueListSize >= lCabinCapacity);
00130
00131     // Copy the EMSR sorted values to the BPV.
00132     stdair::EmsrValueList_T::const_iterator itCurrentValue =
lEmsrValueList.begin();
00133     for (int j = 0; j < lCabinCapacity; ++j, ++itCurrentValue) {
00134         const double lBidPrice = *itCurrentValue;
00135         lBidPriceVector.push_back (lBidPrice);
00136     }
00137     lEmsrValueList.clear();
00138 }

```

```

00151
00152 // Build the protection levels and booking limits.
00153 if (lVirtualClassList.size() > 1) {
00154     int lCapacityIndex = 0;
00155     for (stdair::VirtualClassList_T::iterator itVC = lVirtualClassList.begin()
;
00156         itVC != lVirtualClassList.end();) {
00157         stdair::VirtualClassStruct& lCurrentVC = *itVC;
00158         if (itVC != lVirtualClassList.end()) {
00159             ++itVC;
00160         }
00161         stdair::VirtualClassStruct& lNextVC = *itVC;
00162         const stdair::Yield_T lNextYield = lNextVC.getYield();
00163         while ((lCapacityIndex < lCabinCapacity)
00164             && (lBidPriceVector.at(lCapacityIndex) > lNextYield)) {
00165             ++lCapacityIndex;
00166         }
00167         lCurrentVC.setCumulatedProtection (lCapacityIndex);
00168         lNextVC.setCumulatedBookingLimit (lCapacity - lCapacityIndex);
00169     }
00170 }
00171 }
00172
00173 }

```

26.39 rmol/bom/Emsr.hpp File Reference

```
#include <rmol/RMOL_Types.hpp>
```

Classes

- class [RMOL::Emsr](#)

Namespaces

- namespace [stdair](#)
- namespace [RMOL](#)

26.40 Emsr.hpp

```

00001 #ifndef __RMOL_EMSR_HPP
00002 #define __RMOL_EMSR_HPP
00003
00004 // //////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////
00007 // RMOL
00008 #include <rmol/RMOL_Types.hpp>
00009
00011 namespace stdair {
00012     class LegCabin;
00013 }
00014
00015 namespace RMOL {
00016
00018     class Emsr {
00019     public:
00031         static void heuristicOptimisationByEmsr (
stdair::LegCabin&);
00032
00037         static void heuristicOptimisationByEmsrA (
stdair::LegCabin&);
00038
00043         static void heuristicOptimisationByEmsrB (
stdair::LegCabin&);
00044
00045     };
00046 }
00047 #endif // __RMOL_EMSR_HPP

```

26.41 rmol/bom/EmsrUtils.cpp File Reference

```
#include <cassert>
#include <cmath>
#include <boost/math/distributions/normal.hpp>
#include <stdair/stdair_maths_types.hpp>
#include <stdair/bom/VirtualClassStruct.hpp>
#include <rmol/bom/EmsrUtils.hpp>
#include <rmol/basic/BasConst_General.hpp>
```

Namespaces

- namespace [RMOL](#)

26.42 EmsrUtils.cpp

```
00001 // //////////////////////////////////////
00002 // Import section
00003 // //////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 #include <cmath>
00007 // Boost Math
00008 #include <boost/math/distributions/normal.hpp>
00009 // StdAir
00010 #include <stdair/stdair_maths_types.hpp>
00011 #include <stdair/bom/VirtualClassStruct.hpp>
00012 // RMOL
00013 #include <rmol/bom/EmsrUtils.hpp>
00014 #include <rmol/basic/BasConst_General.hpp>
00015
00016 namespace RMOL {
00017 // //////////////////////////////////////
00018 void EmsrUtils::computeAggregatedVirtualClass
00019 (stdair::VirtualClassStruct& ioAggregatedVirtualClass,
00020  stdair::VirtualClassStruct& ioCurrentVirtualClass) {
00021     // Retrieve the demand mean, demand standard deviation and average
00022     // yield of the classes/buckets.
00023     const stdair::MeanValue_T lAggregatedMean =
00024         ioAggregatedVirtualClass.getMean();
00025     const stdair::MeanValue_T lCurrentMean = ioCurrentVirtualClass.getMean();
00026     const stdair::StdDevValue_T lAggregatedSD =
00027         ioAggregatedVirtualClass.getStdDev();
00028     const stdair::StdDevValue_T lCurrentSD = ioCurrentVirtualClass.getStdDev();
00029     const stdair::Yield_T lAggregatedYield =
00030         ioAggregatedVirtualClass.getYield();
00031     const stdair::Yield_T lCurrentYield = ioCurrentVirtualClass.getYield();
00032
00033     // Compute the new demand mean, new demand standard deviation and
00034     // new average yield for the new aggregated class/bucket.
00035     const stdair::MeanValue_T lNewMean = lAggregatedMean + lCurrentMean;
00036     const stdair::StdDevValue_T lNewSD =
00037         sqrt (lAggregatedSD*lAggregatedSD + lCurrentSD*lCurrentSD);
00038     stdair::Yield_T lNewYield = lCurrentYield;
00039     if (lNewMean > 0) {
00040         lNewYield = (lAggregatedYield*lAggregatedMean +
00041                     lCurrentYield*lCurrentMean) / lNewMean;
00042     }
00043     // Set the new yield range for the new aggregated class/bucket.
00044     ioAggregatedVirtualClass.setYield(lNewYield);
00045
00046     // Set the new demand for the new aggregated class/bucket.
00047     ioAggregatedVirtualClass.setMean (lNewMean);
00048     ioAggregatedVirtualClass.setStdDev (lNewSD);
00049 }
00050
00051 // //////////////////////////////////////
00052 const stdair::ProtectionLevel_T EmsrUtils::
00053 computeProtectionLevel (stdair::VirtualClassStruct&
00054 ioAggregatedVirtualClass,
00055                          stdair::VirtualClassStruct& ioNextVirtualClass) {
00056     // Retrieve the mean & standard deviation of the aggregated
00057     // class/bucket and the average yield of all the two
00058     // classes/buckets.
00059     const stdair::MeanValue_T lMean = ioAggregatedVirtualClass.getMean();
00060     const stdair::StdDevValue_T lSD = ioAggregatedVirtualClass.getStdDev();
```

```

00060     const stdair::Yield_T lAggregatedYield = ioAggregatedVirtualClass.getYield()
;
00061     const stdair::Yield_T lNextYield = ioNextVirtualClass.getYield();
00062     assert (lAggregatedYield != 0);
00063
00064     // Compute the yield ratio between the higher bucket and the current one
00065     const double lYieldRatio = lNextYield / lAggregatedYield;
00066
00070     boost::math::normal lNormalDistribution (lMean, lSD);
00071     const stdair::ProtectionLevel_T lProtection =
00072         boost::math::quantile (boost::math::complement (lNormalDistribution,
00073                                                         lYieldRatio));
00074
00075     return lProtection;
00076 }
00077
00078 // //////////////////////////////////////
00079 const double EmsrUtils::
00080 computeEmsrValue (double iCapacity,
00081                  stdair::VirtualClassStruct& ioVirtualClass){
00082     // Retrieve the average yield, mean and standard deviation of the
00083     // demand of the class/bucket.
00084     const stdair::MeanValue_T lMean = ioVirtualClass.getMean();
00085     const stdair::StdDevValue_T lSD = ioVirtualClass.getStdDev();
00086     const stdair::Yield_T lYield = ioVirtualClass.getYield();
00087
00088     // Compute the EMSR value = lYield * Pr (demand >= iCapacity).
00089     boost::math::normal lNormalDistribution (lMean, lSD);
00090     const double emsrValue =
00091         lYield * boost::math::cdf (boost::math::complement (lNormalDistribution,
00092                                                             iCapacity));
00093
00094     return emsrValue;
00095 }
00096 }

```

26.43 rmol/bom/EmsrUtils.hpp File Reference

```
#include <stdair/stdair_inventory_types.hpp>
```

Classes

- class [RMOL::EmsrUtils](#)

Namespaces

- namespace [stdair](#)
- namespace [RMOL](#)

26.44 EmsrUtils.hpp

```

00001 #ifndef __RMOL_EMSRUTILS_HPP
00002 #define __RMOL_EMSRUTILS_HPP
00003
00004 // //////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////
00007 // StdAir
00008 #include <stdair/stdair_inventory_types.hpp>
00009
00010 // Forward declarations.
00011 namespace stdair {
00012     struct VirtualClassStruct;
00013 }
00014
00015 namespace RMOL {
00016
00019     class EmsrUtils {
00020     public:
00023         static void computeAggregatedVirtualClass (
00024             stdair::VirtualClassStruct&,
00025             stdair::VirtualClassStruct&);

```

```

00027     static const stdair::ProtectionLevel_T computeProtectionLevel
        (stdair::VirtualClassStruct&, stdair::VirtualClassStruct&);
00028
00030     static const double computeEmsrValue (double,
        stdair::VirtualClassStruct&);
00031 };
00032 }
00033 #endif // __RMOL_EMSRUTILS_HPP

```

26.45 rmol/bom/GuillotineBlockHelper.cpp File Reference

```

#include <cassert>
#include <stdair/basic/BasConst_Inventory.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/SegmentDate.hpp>
#include <stdair/bom/SegmentCabin.hpp>
#include <stdair/bom/BookingClass.hpp>
#include <stdair/bom/GuillotineBlock.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/bom/GuillotineBlockHelper.hpp>

```

Namespaces

- namespace [RMOL](#)

26.46 GuillotineBlockHelper.cpp

```

00001 // //////////////////////////////////////
00002 // Import section
00003 // //////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 // StdAir
00007 #include <stdair/basic/BasConst_Inventory.hpp>
00008 #include <stdair/bom/BomManager.hpp>
00009 #include <stdair/bom/SegmentDate.hpp>
00010 #include <stdair/bom/SegmentCabin.hpp>
00011 #include <stdair/bom/BookingClass.hpp>
00012 #include <stdair/bom/GuillotineBlock.hpp>
00013 #include <stdair/service/Logger.hpp>
00014 // RMOL
00015 #include <rmol/bom/GuillotineBlockHelper.hpp>
00016
00017 namespace RMOL {
00018     // //////////////////////////////////////
00019     stdair::NbOfSegments_T GuillotineBlockHelper::
00020     getNbOfSegmentAlreadyPassedThisDTD (const
        stdair::GuillotineBlock& iGB,
00021                                         const stdair::DTD_T& iDTD,
00022                                         const stdair::Date_T& iCurrentDate) {
00023         stdair::NbOfSegments_T oNbOfSegments = 0;
00024
00025         // Browse the list of segments and check if it has passed the given DTD.
00026         const stdair::SegmentCabinIndexMap_T& lSCMap=iGB.getSegmentCabinIndexMap();
00027         for (stdair::SegmentCabinIndexMap_T::const_iterator itSC = lSCMap.begin();
00028             itSC != lSCMap.end(); ++itSC) {
00029             const stdair::SegmentCabin* lSC_ptr = itSC->first;
00030             assert (lSC_ptr != NULL);
00031
00032             if (hasPassedThisDTD (*lSC_ptr, iDTD, iCurrentDate) ==
00033                 true) {
00034                 ++oNbOfSegments;
00035             }
00036         }
00037         return oNbOfSegments;
00038     }
00039
00040     // //////////////////////////////////////
00041     bool GuillotineBlockHelper::
00042     hasPassedThisDTD (const stdair::SegmentCabin& iSegmentCabin
        ,

```



```

00043         const stdair::DTD_T& iDTD,
00044         const stdair::Date_T& iCurrentDate) {
00045     // Retrieve the boarding date.
00046     const stdair::SegmentDate& lSegmentDate =
00047         stdair::BomManager::getParent<stdair::SegmentDate> (iSegmentCabin);
00048     const stdair::Date_T& lBoardingDate = lSegmentDate.getBoardingDate();
00049
00050     // Compare the date offset between the boarding date and the current date
00051     // to the DTD.
00052     stdair::DateOffset_T lDateOffset = lBoardingDate - iCurrentDate;
00053     stdair::DTD_T lDateOffsetInDays = lDateOffset.days();
00054     if (iDTD < lDateOffsetInDays) {
00055         return false;
00056     } else {
00057         return true;
00058     }
00059 }
00060 }

```

26.47 rmol/bom/GuillotineBlockHelper.hpp File Reference

```

#include <string>
#include <stdair/stdair_inventory_types.hpp>
#include <stdair/stdair_date_time_types.hpp>

```

Classes

- class [RMOL::GuillotineBlockHelper](#)

Namespaces

- namespace [stdair](#)
- namespace [RMOL](#)

26.48 GuillotineBlockHelper.hpp

```

00001 #ifndef __RMOL_BOM_GUILLOTINEBLOCKHELPER_HPP
00002 #define __RMOL_BOM_GUILLOTINEBLOCKHELPER_HPP
00003
00004 // //////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////
00007 // STL
00008 #include <string>
00009 // StdAir
00010 #include <stdair/stdair_inventory_types.hpp>
00011 #include <stdair/stdair_date_time_types.hpp>
00012
00013 // Forward declarations
00014 namespace stdair {
00015     class GuillotineBlock;
00016     class SegmentCabin;
00017 }
00018
00019 namespace RMOL {
00020
00023     class GuillotineBlockHelper {
00024     public:
00025         // ////////////////////////////////////// Business Methods //////////////////////////////////////
00030         static stdair::NbOfSegments_T getNbOfSegmentAlreadyPassedThisDTD
00031         (const stdair::GuillotineBlock&, const stdair::DTD_T&, const stdair::Date_T&);
00035         static bool hasPassedThisDTD (const stdair::SegmentCabin&,
00036         const stdair::DTD_T&, const stdair::Date_T&);
00037     };
00038
00039 }
00040 #endif // __RMOL_BOM_GUILLOTINEBLOCKHELPER_HPP

```

26.49 rmol/bom/HistoricalBooking.cpp File Reference

```
#include <sstream>
#include <cassert>
#include <iomanip>
#include <iostream>
#include <rmol/bom/HistoricalBooking.hpp>
```

Namespaces

- namespace [RMOL](#)

26.50 HistoricalBooking.cpp

```
00001 ///////////////////////////////////////////////////////////////////
00002 // Import section
00003 ///////////////////////////////////////////////////////////////////
00004 // STL
00005 #include <sstream>
00006 #include <cassert>
00007 #include <iomanip>
00008 #include <iostream>
00009 // RMOL
00010 #include <rmol/bom/HistoricalBooking.hpp>
00011
00012 namespace RMOL {
00013
00014 ///////////////////////////////////////////////////////////////////
00015 HistoricalBooking::HistoricalBooking () :
00016     _numberOfBookings (0.0), _unconstrainedDemand (0.0), _flag (false) {
00017 }
00018
00019 ///////////////////////////////////////////////////////////////////
00020 HistoricalBooking::
00021 HistoricalBooking (const stdair::NbOfBookings_T
00022 iNbOfBookings,
00023                 const stdair::Flag_T iFlag)
00024 : _numberOfBookings (iNbOfBookings),
00025   _unconstrainedDemand (iNbOfBookings), _flag (iFlag) {
00026 }
00027 ///////////////////////////////////////////////////////////////////
00028 HistoricalBooking::HistoricalBooking
00029 (const HistoricalBooking& iHistoricalBooking) :
00030     _numberOfBookings (iHistoricalBooking.getNbOfBookings()),
00031     _unconstrainedDemand (iHistoricalBooking.getUnconstrainedDemand
00032 ()),
00033     _flag (iHistoricalBooking.getFlag()) {
00034 }
00035 ///////////////////////////////////////////////////////////////////
00036 HistoricalBooking::~HistoricalBooking()
00037 {
00038 }
00039 ///////////////////////////////////////////////////////////////////
00040 void HistoricalBooking::setParameters
00041 (const stdair::NbOfBookings_T iNbOfBookings, const stdair::Flag_T iFlag) {
00042     _numberOfBookings = iNbOfBookings;
00043     _unconstrainedDemand = iNbOfBookings;
00044     _flag = iFlag;
00045 }
00046
00047 ///////////////////////////////////////////////////////////////////
00048 const std::string HistoricalBooking::describe()
00049 const {
00050     std::ostringstream ostr;
00051     ostr << "Struct of hitorical booking, unconstrained demand and flag of "
00052         << "censorship for a FlightDate/Class.";
00053     return ostr.str();
00054 }
00055
00056 ///////////////////////////////////////////////////////////////////
00057 void HistoricalBooking::toStream (std::ostream&
00058 ioOut) const {
```

```

00058     const stdair::NbOfBookings_T bj = getNbOfBookings();
00059     const stdair::NbOfBookings_T uj = getUnconstrainedDemand
00060 ();
00060     const stdair::Flag_T fj = getFlag();
00061     ioOut << std::fixed << std::setprecision (2)
00062         << bj << " "; " << uj << " "; " << fj << std::endl;
00063 }
00064
00065 // //////////////////////////////////////
00066 void HistoricalBooking::display () const {
00067     toStream (std::cout);
00068 }
00069 }

```

26.51 rmol/bom/HistoricalBooking.hpp File Reference

```

#include <stdair/stdair_basic_types.hpp>
#include <stdair/basic/StructAbstract.hpp>

```

Classes

- struct [RMOL::HistoricalBooking](#)
Structure keeping track, for a given class, of the number of historical bookings and of the censorship flag.

Namespaces

- namespace [RMOL](#)

26.52 HistoricalBooking.hpp

```

00001 #ifndef __RMOL_BOM_HISTORICALBOOKING_HPP
00002 #define __RMOL_BOM_HISTORICALBOOKING_HPP
00003
00004 // //////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////
00007 // StdAir
00008 #include <stdair/stdair_basic_types.hpp>
00009 #include <stdair/basic/StructAbstract.hpp>
00010
00011 namespace RMOL {
00012
00017     struct HistoricalBooking : public stdair::StructAbstract {
00018
00019     public:
00020         // ////////////////////////////////////// Getters //////////////////////////////////////
00022         const stdair::NbOfBookings_T& getNbOfBookings() const {
00023             return _numberOfBookings;
00024         }
00026         const stdair::NbOfBookings_T& getUnconstrainedDemand(
00027 ) const {
00028             return _unconstrainedDemand;
00029         }
00031         const stdair::Flag_T& getFlag() const {
00032             return _flag;
00033         }
00034
00035     public:
00036         // ////////////////////////////////////// Setters //////////////////////////////////////
00038         void setUnconstrainedDemand (const
stdair::NbOfBookings_T& iDemand) {
00039             _unconstrainedDemand = iDemand;
00040         }
00043         void setParameters (const stdair::NbOfBookings_T, const
stdair::Flag_T);
00044
00045     public:
00046         // ////////////////////////////////////// Display Methods //////////////////////////////////////
00052         void toStream (std::ostream& ioOut) const;
00053
00057         const std::string describe() const;

```

```

00058
00062     void display () const;
00063
00064     public:
00065         // ////////// Constructors and destructor. //////////
00069         HistoricalBooking (const stdair::NbOfBookings_T, const
stdair::Flag_T);
00073         HistoricalBooking();
00077         HistoricalBooking (const HistoricalBooking
&);
00078
00082         virtual ~HistoricalBooking();
00083
00084     private:
00085         // ////////// Attributes //////////
00089         stdair::NbOfBookings_T _numberOfBookings;
00090
00094         stdair::NbOfBookings_T _unconstrainedDemand;
00095
00099         stdair::Flag_T _flag;
00100     };
00101 }
00102 #endif // __RMOL_BOM_HISTORICALBOOKING_HPP

```

26.53 rmol/bom/HistoricalBookingHolder.cpp File Reference

```

#include <sstream>
#include <iostream>
#include <iomanip>
#include <cmath>
#include <cassert>
#include <stdair/service/Logger.hpp>
#include <rmol/bom/HistoricalBooking.hpp>
#include <rmol/bom/HistoricalBookingHolder.hpp>

```

Namespaces

- namespace [RMOL](#)

26.54 HistoricalBookingHolder.cpp

```

00001 // //////////////////////////////////////
00002 // Import section
00003 // //////////////////////////////////////
00004 // STL
00005 #include <sstream>
00006 #include <iostream>
00007 #include <iomanip>
00008 #include <cmath>
00009 #include <cassert>
00010 // StdAir
00011 #include <stdair/service/Logger.hpp>
00012 // RMOL
00013 #include <rmol/bom/HistoricalBooking.hpp>
00014 #include <rmol/bom/HistoricalBookingHolder.hpp>
00015 >
00016 namespace RMOL {
00017
00018     // //////////////////////////////////////
00019     HistoricalBookingHolder::HistoricalBookingHolder
00020     () {
00021     }
00022
00023     // //////////////////////////////////////
00024     HistoricalBookingHolder::~HistoricalBookingHolder
00025     () {
00026         _historicalBookingVector.clear();
00027     }
00028
00029     // //////////////////////////////////////
00030     const short HistoricalBookingHolder::getNbOfFlights
00031     () const {

```

```

00029     return _historicalBookingVector.size();
00030 }
00031
00032 // //////////////////////////////////////
00033 const short HistoricalBookingHolder::getNbOfUncensoredData
00034 () const {
00035     short lResult = 0;
00036     const short lSize = _historicalBookingVector.size();
00037     for (short ite = 0; ite < lSize; ++ite) {
00038         const stdair::Flag_T lFlag = _historicalBookingVector.at(ite).getFlag ();
00039         if (lFlag == false) {
00040             ++ lResult;
00041         }
00042     }
00043     return lResult;
00044 }
00045
00046 // //////////////////////////////////////
00047 const stdair::NbOfBookings_T HistoricalBookingHolder::
00048 getNbOfUncensoredBookings () const {
00049     stdair::NbOfBookings_T lResult = 0;
00050     const short lSize = _historicalBookingVector.size();
00051     for (short ite = 0; ite < lSize; ++ite) {
00052         const HistoricalBooking& lHistorialBooking =
00053             _historicalBookingVector.at (ite);
00054         const stdair::Flag_T lFlag = lHistorialBooking.getFlag ();
00055         if (lFlag == false) {
00056             const stdair::NbOfBookings_T& lBooking =
00057                 lHistorialBooking.getNbOfBookings ();
00058             lResult += lBooking;
00059         }
00060     }
00061     return lResult;
00062 }
00063
00064 // //////////////////////////////////////
00065 const double HistoricalBookingHolder::
00066 getUncensoredStandardDeviation (const double&
00067 iMeanOfUncensoredBookings,
00068                                 const short iNbOfUncensoredData) const {
00069     double lResult = 0;
00070     const short lSize = _historicalBookingVector.size();
00071     for (short ite = 0; ite < lSize; ++ite) {
00072         const stdair::Flag_T lFlag = _historicalBookingVector.at(ite).getFlag ();
00073         if (lFlag == false) {
00074             const HistoricalBooking& lHistorialBooking =
00075                 _historicalBookingVector.at (ite);
00076             const stdair::NbOfBookings_T& lBooking =
00077                 lHistorialBooking.getNbOfBookings ();
00078             lResult += (lBooking - iMeanOfUncensoredBookings)
00079                 * (lBooking - iMeanOfUncensoredBookings);
00080         }
00081     }
00082     lResult /= (iNbOfUncensoredData - 1);
00083     lResult = sqrt (lResult);
00084     return lResult;
00085 }
00086
00087 // //////////////////////////////////////
00088 const double HistoricalBookingHolder::getDemandMean
00089 () const {
00090     double lResult = 0;
00091     const short lSize = _historicalBookingVector.size();
00092     for (short ite = 0; ite < lSize; ++ite) {
00093         const HistoricalBooking& lHistorialBooking =
00094             _historicalBookingVector.at(ite);
00095         const stdair::NbOfBookings_T& lDemand =
00096             lHistorialBooking.getUnconstrainedDemand ();
00097         lResult += static_cast<double>(lDemand);
00098     }
00099     lResult /= lSize;
00100     return lResult;
00101 }
00102

```

```

00113
00114 ///////////////////////////////////////////////////////////////////
00115 const double HistoricalBookingHolder::getStandardDeviation
00116 (const double iDemandMean) const {
00117     double lResult = 0;
00118     const short lSize = _historicalBookingVector.size();
00119
00120     for (short ite = 0; ite < lSize; ++ite) {
00121         const HistoricalBooking& lHistorialBooking =
00122             _historicalBookingVector.at(ite);
00123
00124         const stdair::NbOfBookings_T& lDemand =
00125             lHistorialBooking.getUnconstrainedDemand ();
00126
00127         const double lDoubleDemand = static_cast<double> (lDemand);
00128         lResult += (lDoubleDemand - iDemandMean) * (lDoubleDemand - iDemandMean);
00129     }
00130
00131     lResult /= (lSize - 1);
00132
00133     lResult = sqrt (lResult);
00134
00135     return lResult;
00136 }
00137
00138 ///////////////////////////////////////////////////////////////////
00139 const std::vector<bool> HistoricalBookingHolder::
00140 getListOfToBeUnconstrainedFlags () const {
00141     std::vector<bool> lResult;
00142     const short lSize = _historicalBookingVector.size();
00143
00144     for (short ite = 0; ite < lSize; ++ite) {
00145         const HistoricalBooking& lHistorialBooking =
00146             _historicalBookingVector.at(ite);
00147         const stdair::Flag_T lFlag = lHistorialBooking.getFlag ();
00148         if (lFlag == true) {
00149             lResult.push_back(true);
00150         }
00151         else {
00152             lResult.push_back(false);
00153         }
00154     }
00155
00156     return lResult;
00157 }
00158
00159 ///////////////////////////////////////////////////////////////////
00160 const stdair::NbOfBookings_T& HistoricalBookingHolder::
00161 getHistoricalBooking (const short i) const {
00162     const HistoricalBooking& lHistorialBooking =
00163         _historicalBookingVector.at(i);
00164     return lHistorialBooking.getNbOfBookings();
00165 }
00166
00167 ///////////////////////////////////////////////////////////////////
00168 const stdair::NbOfBookings_T& HistoricalBookingHolder::
00169 getUnconstrainedDemand (const short i) const {
00170     const HistoricalBooking& lHistorialBooking =
00171         _historicalBookingVector.at(i);
00172     return lHistorialBooking.getUnconstrainedDemand();
00173 }
00174
00175 ///////////////////////////////////////////////////////////////////
00176 const stdair::Flag_T& HistoricalBookingHolder::
00177 getCensorshipFlag (const short i) const {
00178     const HistoricalBooking& lHistorialBooking =
00179         _historicalBookingVector.at(i);
00180     return lHistorialBooking.getFlag();
00181 }
00182
00183 ///////////////////////////////////////////////////////////////////
00184 void HistoricalBookingHolder::setUnconstrainedDemand
00185 (const stdair::NbOfBookings_T& iExpectedDemand, const short i) {
00186     _historicalBookingVector.at(i).setUnconstrainedDemand(iExpectedDemand);
00187 }
00188
00189 ///////////////////////////////////////////////////////////////////
00190 const stdair::NbOfBookings_T HistoricalBookingHolder::calculateExpectedDemand
00191 (const double iMean, const double iSD,
00192  const short i, const stdair::NbOfBookings_T iDemand) const {
00193
00194     const HistoricalBooking lHistorialBooking =
00195         _historicalBookingVector.at(i);
00196     const double lBooking =
00197         static_cast<double> (lHistorialBooking.getNbOfBookings())
00198 ;
    double e, d1, d2;

```

```

00199
00200     e = - (lBooking - iMean) * (lBooking - iMean) * 0.625 / (iSD * iSD);
00201     //STDAIR_LOG_DEBUG ("e: " << e);
00202     e = exp (e);
00203     //STDAIR_LOG_DEBUG ("e: " << e);
00204
00205     double s = sqrt (1 - e);
00206     //STDAIR_LOG_DEBUG ("s: " << s);
00207
00208     if (lBooking >= iMean) {
00209         if (e < 0.01) {
00210             return iDemand;
00211         }
00212         d1 = 0.5 * (1 - s);
00213     }
00214     else {
00215         d1 = 0.5 * (1 + s);
00216     }
00217     //STDAIR_LOG_DEBUG ("d1: " << d1);
00218
00219     e = - (lBooking - iMean) * (lBooking - iMean) * 0.5 / (iSD * iSD);
00220     e = exp (e);
00221     d2 = e * iSD / sqrt (2 * 3.14159265);
00222     //STDAIR_LOG_DEBUG ("d2: " << d2);
00223
00224     if (d1 == 0) {
00225         return iDemand;
00226     }
00227
00228     const stdair::NbOfBookings_T lDemand =
00229         static_cast<stdair::NbOfBookings_T> (iMean + d2/d1);
00230
00231     return lDemand;
00232 }
00233
00234 // //////////////////////////////////////
00235 void HistoricalBookingHolder::addHistoricalBooking
00236 (const HistoricalBooking& iHistoricalBooking) {
00237     _historicalBookingVector.push_back(iHistoricalBooking);
00238 }
00239
00240 // //////////////////////////////////////
00241 void HistoricalBookingHolder::toStream (
00242     std::ostream& ioOut) const {
00243     const short lSize = _historicalBookingVector.size();
00244
00245     ioOut << "Historical Booking; Unconstrained Demand; Flag" << std::endl;
00246
00247     for (short ite = 0; ite < lSize; ++ite) {
00248         const HistoricalBooking& lHistoricalBooking =
00249             _historicalBookingVector.at(ite);
00250
00251         const stdair::NbOfBookings_T& lBooking =
00252             lHistoricalBooking.getNbOfBookings();
00253
00254         const stdair::NbOfBookings_T& lDemand =
00255             lHistoricalBooking.getUnconstrainedDemand();
00256
00257         const stdair::Flag_T lFlag = lHistoricalBooking.getFlag();
00258
00259         ioOut << lBooking << " "
00260             << lDemand << " "
00261             << lFlag << std::endl;
00262     }
00263
00264 // //////////////////////////////////////
00265 const std::string HistoricalBookingHolder::describe
00266 () const {
00267     std::ostringstream ostr;
00268     ostr << "Holder of HistoricalBooking structs.";
00269     return ostr.str();
00270 }
00271
00272 // //////////////////////////////////////
00273 void HistoricalBookingHolder::display() const
00274 {
00275     toStream (std::cout);
00276 }

```

26.55 rmol/bom/HistoricalBookingHolder.hpp File Reference

```
#include <iostream>
#include <vector>
#include <stdair/stdair_basic_types.hpp>
#include <stdair/basic/StructAbstract.hpp>
```

Classes

- struct [RMOL::HistoricalBookingHolder](#)

Namespaces

- namespace [RMOL](#)

Typedefs

- typedef std::vector
 < HistoricalBooking > [RMOL::HistoricalBookingVector_T](#)

26.56 HistoricalBookingHolder.hpp

```
00001 #ifndef __RMOL_BOM_HISTORICALBOOKINGHOLDER_HPP
00002 #define __RMOL_BOM_HISTORICALBOOKINGHOLDER_HPP
00003
00004 // //////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////
00007 // STL
00008 #include <iostream>
00009 #include <vector>
00010 // StdAir
00011 #include <stdair/stdair_basic_types.hpp>
00012 #include <stdair/basic/StructAbstract.hpp>
00013
00014 namespace RMOL {
00016     struct HistoricalBooking;
00017
00019     typedef std::vector<HistoricalBooking> HistoricalBookingVector_T
00020 ;
00023     struct HistoricalBookingHolder : public
stdair::StructAbstract {
00024
00025     public:
00026         // ///// Getters /////
00028         const short getNbOfFlights () const;
00029
00031         const short getNbOfUncensoredData () const;
00032
00034         const stdair::NbOfBookings_T getNbOfUncensoredBookings
() const;
00035
00037         const double getUncensoredStandardDeviation
(const double& iMeanOfUncensoredBookings,
00038         const short iNbOfUncensoredData) const;
00039
00042         const double getDemandMean () const;
00043
00045         const double getStandardDeviation (const double) const;
00046
00048         const std::vector<bool> getListOfToBeUnconstrainedFlags
() const;
00049
00051         const stdair::NbOfBookings_T& getHistoricalBooking (
const short i) const;
00052
00054         const stdair::NbOfBookings_T& getUnconstrainedDemand
(const short i) const;
00055
00057         const stdair::Flag_T& getCensorshipFlag (const short i)
```



```

    const;
00058
00060     const stdair::NbOfBookings_T& getUnconstrainedDemandOnFirstElement
    () const {
00061         return getUnconstrainedDemand (0);
00062     }
00063
00065     const stdair::NbOfBookings_T calculateExpectedDemand
    (const double,
00066                                     const double,
00067                                     const short,
00068                                     const stdair::NbOfBookings_T) const
    ;
00069
00071     void setUnconstrainedDemand (const
    stdair::NbOfBookings_T& iExpectedDemand,
00072                                     const short i);
00073
00075     void addHistoricalBooking (const HistoricalBooking
    & iHistoricalBooking);
00076
00080     void toStream (std::ostream& ioOut) const;
00081
00082     // /////////// Display Methods ///////////
00084     const std::string describe() const;
00085
00087     void display () const;
00088
00090     virtual ~HistoricalBookingHolder();
00091
00092 public:
00095     HistoricalBookingHolder ();
00096
00097 private:
00099     HistoricalBookingVector_T _historicalBookingVector
    ;
00100
00101 protected:
00102 };
00103 }
00104 #endif // __RMOL_BOM_HISTORICALBOOKINGHOLDER_HPP
00105

```

26.57 rmol/bom/MCOptimiser.cpp File Reference

```

#include <cassert>
#include <string>
#include <sstream>
#include <algorithm>
#include <cmath>
#include <stdair/stdair_basic_types.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/LegCabin.hpp>
#include <stdair/bom/SegmentCabin.hpp>
#include <stdair/bom/BookingClass.hpp>
#include <stdair/bom/VirtualClassStruct.hpp>
#include <stdair/service/Logger.hpp>
#include <stdair/basic/RandomGeneration.hpp>
#include <stdair/basic/BasConst_General.hpp>
#include <rmol/bom/MCOptimiser.hpp>

```

Namespaces

- namespace [RMOL](#)

26.58 MCOptimiser.cpp

```

00001 // //////////////////////////////////////
00002 // Import section

```

```

00003 // //////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 #include <string>
00007 #include <sstream>
00008 #include <algorithm>
00009 #include <cmath>
00010 // StdAir
00011 #include <stdair/stdair_basic_types.hpp>
00012 #include <stdair/bom/BomManager.hpp>
00013 #include <stdair/bom/LegCabin.hpp>
00014 #include <stdair/bom/SegmentCabin.hpp>
00015 #include <stdair/bom/BookingClass.hpp>
00016 #include <stdair/bom/VirtualClassStruct.hpp>
00017 #include <stdair/service/Logger.hpp>
00018
00019 #include <stdair/basic/RandomGeneration.hpp>
00020 #include <stdair/basic/BasConst_General.hpp>
00021 // RMOL
00022 #include <rmol/bom/MCOptimiser.hpp>
00023
00024 namespace RMOL {
00025
00026 // // //////////////////////////////////////
00027 void MCOptimiser::
00028 optimalOptimisationByMCIntegration (
stdair::LegCabin& ioLegCabin) {
00029 // Retrieve the segment-cabin
00030 const stdair::SegmentCabinList_T lSegmentCabinList =
00031     stdair::BomManager::getList<stdair::SegmentCabin> (ioLegCabin);
00032     stdair::SegmentCabinList_T::const_iterator itSC = lSegmentCabinList.begin()
;
00033     assert (itSC != lSegmentCabinList.end());
00034     const stdair::SegmentCabin* lSegmentCabin_ptr = *itSC;
00035     assert (lSegmentCabin_ptr != NULL);
00036
00037 // Retrieve the class list.
00038 const stdair::BookingClassList_T lBookingClassList =
00039     stdair::BomManager::getList<stdair::BookingClass> (*lSegmentCabin_ptr);
00040
00041 // Retrieve the remaining cabin capacity.
00042 const stdair::Availability_T& lCap = ioLegCabin.getAvailabilityPool();
00043 const int lCapacity = static_cast<const int> (lCap);
00044 const stdair::UnsignedIndex_T lCapacityIndex =
00045     static_cast<const stdair::UnsignedIndex_T> ((lCapacity+abs(lCapacity))/2)
;
00046
00047 // Retrieve the virtual class list.
00048 stdair::VirtualClassList_T& lVCList = ioLegCabin.getVirtualClassList();
00049
00050 // Parse the virtual class list and compute the protection levels.
00051 stdair::VirtualClassList_T::iterator itCurrentVC = lVCList.begin();
00052 assert (itCurrentVC != lVCList.end());
00053 stdair::VirtualClassList_T::iterator itNextVC = itCurrentVC; ++itNextVC;
00054
00055 // Initialise the partial sum holder with the demand sample of the first
// virtual class.
00056 stdair::VirtualClassStruct& lFirstVC = *itCurrentVC;
00057 stdair::GeneratedDemandVector_T lPartialSumHolder =
00058     lFirstVC.getGeneratedDemandVector();
00059
00060 // Initialise the booking limit for the first class, which is equal to
// the remaining capacity.
00061 lFirstVC.setCumulatedBookingLimit (lCap);
00062
00063 // Initialise bid price vector with the first element (index 0) equal to
// the highest yield.
00064 ioLegCabin.emptyBidPriceVector();
00065 stdair::BidPriceVector_T& lBPV = ioLegCabin.getBidPriceVector();
00066 //const stdair::Yield_T& y1 = lFirstVC.getYield ();
00067 //lBPV.push_back (y1);
00068 stdair::UnsignedIndex_T idx = 1;
00069
00070 for (; itNextVC != lVCList.end(); ++itCurrentVC, ++itNextVC) {
00071 // Get the yields of the two classes.
00072 stdair::VirtualClassStruct& lCurrentVC = *itCurrentVC;
00073 stdair::VirtualClassStruct& lNextVC = *itNextVC;
00074 const stdair::Yield_T& yj = lCurrentVC.getYield ();
00075 const stdair::Yield_T& yj1 = lNextVC.getYield ();
00076
00077 // Consistency check: the yield/price of a higher class/bucket
// (with the j index lower) must be higher.
00078 assert (yj > yj1);
00079
00080 // Sort the partial sum holder.
00081 std::sort (lPartialSumHolder.begin(), lPartialSumHolder.end());
00082 const stdair::UnsignedIndex_T K = lPartialSumHolder.size ();

```

```

00087
00088 // Compute the optimal index  $lj = \text{floor} \{ [y(j)-y(j+1)]/y(j) \cdot K \}$ 
00089 const double ljdoube = std::floor (K * (yj - yj1) / yj);
00090 stdair::UnsignedIndex_T lj =
00091     static_cast<stdair::UnsignedIndex_T> (ljdoube);
00092
00093 // Consistency check.
00094 assert (lj >= 1 && lj < K);
00095
00096 // The optimal protection:  $p(j) = 1/2 [S(j,lj) + S(j, lj+1)]$ 
00097 const double sj1 = lPartialSumHolder.at (lj - 1);
00098 const double sjlp1 = lPartialSumHolder.at (lj + 1 - 1);
00099 const double pj = (sj1 + sjlp1) / 2;
00100
00101 // Set the cumulated protection level for the current class.
00102 lCurrentVC.setCumulatedProtection (pj);
00103 // Set the cumulated booking limit for the next class.
00104 lNextVC.setCumulatedBookingLimit (lCap - pj);
00105
00106 const stdair::UnsignedIndex_T pjint = static_cast<const int> (pj);
00107 stdair::GeneratedDemandVector_T::iterator itLowerBound =
00108     lPartialSumHolder.begin();
00109 for (; idx <= pjint && idx <= lCapacityIndex; ++idx) {
00110     itLowerBound =
00111         std::lower_bound (itLowerBound, lPartialSumHolder.end(), idx);
00112     const stdair::UnsignedIndex_T pos =
00113         itLowerBound - lPartialSumHolder.begin();
00114     const stdair::BidPrice_T lBP = yj * (K - pos) / K;
00115     lBPV.push_back (lBP);
00116 }
00117
00118 // Update the partial sum holder.
00119 const stdair::GeneratedDemandVector_T& lNextPSH =
00120     lNextVC.getGeneratedDemandVector();
00121 assert (K <= lNextPSH.size());
00122 for (stdair::UnsignedIndex_T i = 0; i < K - lj; ++i) {
00123     lPartialSumHolder.at(i) = lPartialSumHolder.at(i + lj) + lNextPSH.at(i)
00124 ;
00125 }
00126 lPartialSumHolder.resize (K - lj);
00127 }
00128
00129 stdair::VirtualClassStruct& lLastVC = *itCurrentVC;
00130 const stdair::Yield_T& yn = lLastVC.getYield();
00131 stdair::GeneratedDemandVector_T::iterator itLowerBound =
00132     lPartialSumHolder.begin();
00133 for (; idx <= lCapacityIndex; ++idx) {
00134     itLowerBound =
00135         std::lower_bound (itLowerBound, lPartialSumHolder.end(), idx);
00136     const stdair::UnsignedIndex_T pos =
00137         itLowerBound - lPartialSumHolder.begin();
00138     const stdair::UnsignedIndex_T K = lPartialSumHolder.size();
00139     const stdair::BidPrice_T lBP = yn * (K - pos) / K;
00140     lBPV.push_back (lBP);
00141 }
00142 }
00143
00144 // //////////////////////////////////////
00145 stdair::GeneratedDemandVector_T MCOptimiser::
00146 generateDemandVector (const stdair::MeanValue_T& iMean,
00147                     const stdair::StdDevValue_T& iStdDev,
00148                     const unsigned int& K) {
00149     stdair::GeneratedDemandVector_T oDemandVector;
00150     if (iStdDev > 0) {
00151         stdair::RandomGeneration lGenerator (stdair::DEFAULT_RANDOM_SEED);
00152         for (unsigned int i = 0; i < K; ++i) {
00153             stdair::RealNumber_T lDemandSample =
00154                 lGenerator.generateNormal (iMean, iStdDev);
00155             oDemandVector.push_back (lDemandSample);
00156         }
00157     } else {
00158         for (unsigned int i = 0; i < K; ++i) {
00159             oDemandVector.push_back (iMean);
00160         }
00161     }
00162     return oDemandVector;
00163 }
00164
00165 // //////////////////////////////////////
00166 void MCOptimiser::
00167 optimisationByMCIntegration (stdair::LegCabin&
00168 ioLegCabin) {
00169     // Number of MC samples
00170     unsigned int K = 100000;
00171     const stdair::YieldLevelDemandMap_T& lYieldDemandMap =

```

```

00180     ioLegCabin.getYieldLevelDemandMap();
00181     assert (!lYieldDemandMap.empty());
00182
00183     std::ostringstream oStr;
00184     oStr << "Yield list ";
00185     for (stdair::YieldLevelDemandMap_T::const_iterator itYD =
00186         lYieldDemandMap.begin();
00187         itYD != lYieldDemandMap.end(); ++itYD) {
00188         const stdair::Yield_T& y = itYD->first;
00189         oStr << y << " ";
00190     }
00191
00192     STDAIR_LOG_DEBUG (oStr.str());
00193     ioLegCabin.emptyBidPriceVector();
00194     stdair::BidPriceVector_T& lBidPriceVector =
00195         ioLegCabin.getBidPriceVector();
00196     const stdair::Availability_T& lAvailabilityPool =
00197         ioLegCabin.getAvailabilityPool();
00198     // Initialise the minimal bid price to 1.0 (just to avoid problems
00199     // of division by zero).
00200     const stdair::BidPrice_T& lMinBP = 1.0;
00201
00202     stdair::YieldLevelDemandMap_T::const_reverse_iterator itCurrentYD =
00203         lYieldDemandMap.rbegin();
00204     stdair::YieldLevelDemandMap_T::const_reverse_iterator itNextYD =
00205         itCurrentYD;
00206     ++itNextYD;
00207
00208     // Initialise the partial sum holder
00209     stdair::MeanStdDevPair_T lMeanStdDevPair = itCurrentYD->second;
00210     stdair::GeneratedDemandVector_T lPartialSumHolder =
00211         generateDemandVector(lMeanStdDevPair.first,
00212             lMeanStdDevPair.second, K);
00213
00214     stdair::UnsignedIndex_T idx = 1;
00215     for (; itNextYD!=lYieldDemandMap.rend(); ++itCurrentYD, ++itNextYD) {
00216         const stdair::Yield_T& yj = itCurrentYD->first;
00217         const stdair::Yield_T& yj1 = itNextYD->first;
00218         // Consistency check: the yield/price of a higher class/bucket
00219         // (with the j index lower) must be higher.
00220         assert (yj > yj1);
00221         // Sort the partial sum holder.
00222         std::sort (lPartialSumHolder.begin(), lPartialSumHolder.end());
00223         // STDAIR_LOG_DEBUG ("Partial sums : max = " << lPartialSumHolder.back()
00224         // << " min = " << lPartialSumHolder.front());
00225         K = lPartialSumHolder.size ();
00226         // Compute the optimal index lj = floor {[y(j)-y(j+1)]/y(j) . K}
00227         const double ljdoube = std::floor (K * (yj - yj1) / yj);
00228         stdair::UnsignedIndex_T lj =
00229             static_cast<stdair::UnsignedIndex_T> (ljdoube);
00230         // Consistency check.
00231         assert (lj >= 1 && lj < K);
00232         // The optimal protection: p(j) = 1/2 [S(j,lj) + S(j, lj+1)]
00233         const double sjl = lPartialSumHolder.at (lj - 1);
00234         const double sjlp1 = lPartialSumHolder.at (lj + 1 - 1);
00235         const double pj = (sjl + sjlp1) / 2;
00236         const stdair::UnsignedIndex_T pjint = static_cast<const int> (pj);
00237         stdair::GeneratedDemandVector_T::iterator itLowerBound =
00238             lPartialSumHolder.begin();
00239         for (; idx <= pjint && idx <= lAvailabilityPool; ++idx) {
00240             itLowerBound =
00241                 std::lower_bound (itLowerBound, lPartialSumHolder.end(), idx);
00242             const stdair::UnsignedIndex_T pos =
00243                 itLowerBound - lPartialSumHolder.begin();
00244
00245             const stdair::BidPrice_T lBP = yj * (K - pos) / K;
00246             lBidPriceVector.push_back (lBP);
00247         }
00248         // Update the partial sum holder.
00249         lMeanStdDevPair = itNextYD->second;
00250         const stdair::GeneratedDemandVector_T& lNextDV =
00251             generateDemandVector (lMeanStdDevPair.first,
00252                 lMeanStdDevPair.second, K - lj);
00253         for (stdair::UnsignedIndex_T i = 0; i < K - lj; ++i) {
00254             lPartialSumHolder.at(i) = lPartialSumHolder.at(i + lj) + lNextDV.at(i);
00255         }
00256         lPartialSumHolder.resize (K - lj);
00257     }
00258     // STDAIR_LOG_DEBUG ("Partial sums : max = " << lPartialSumHolder.back()
00259     // << " min = " << lPartialSumHolder.front());
00260
00261     std::sort (lPartialSumHolder.begin(), lPartialSumHolder.end());
00262     const stdair::Yield_T& yn = itCurrentYD->first;
00263     stdair::GeneratedDemandVector_T::iterator itLowerBound =
00264         lPartialSumHolder.begin();
00265     K = lPartialSumHolder.size();
00266

```

```

00277     bool lMinBPReached = false;
00278     for (; idx <= lAvailabilityPool; ++idx) {
00279         itLowerBound =
00280             std::lower_bound (itLowerBound, lPartialSumHolder.end(), idx);
00281
00282         if (!lMinBPReached) {
00283             const stdair::UnsignedIndex_T pos =
00284                 itLowerBound - lPartialSumHolder.begin();
00285             stdair::BidPrice_T lBP = yn * (K - pos) / K;
00286
00287             if (lBP < lMinBP) {
00288                 lBP = lMinBP; lMinBPReached = true;
00289             }
00290
00291             lBidPriceVector.push_back (lBP);
00292
00293         } else {
00294             lBidPriceVector.push_back (lMinBP);
00295         }
00296     }
00297
00298     // Updating the bid price values
00299     ioLegCabin.updatePreviousBidPrice();
00300     ioLegCabin.setCurrentBidPrice (lBidPriceVector.back());
00301
00302     // Compute and display the bid price variation after optimisation
00303     const stdair::BidPrice_T lPreviousBP = ioLegCabin.getPreviousBidPrice();
00304     stdair::BidPrice_T lNewBP = ioLegCabin.getCurrentBidPrice();
00305     // Check
00306     assert (lPreviousBP != 0);
00307     stdair::BidPrice_T lBidPriceDelta = lNewBP - lPreviousBP;
00308
00309     double lBidPriceVariation = 100*lBidPriceDelta/lPreviousBP;
00310
00311     STDAIR_LOG_DEBUG ("Bid price: previous value " << lPreviousBP
00312                     << ", new value " << lNewBP
00313                     << ", variation " << lBidPriceVariation << " %"
00314                     << ", BPV size " << lBidPriceVector.size());
00315 }
00316
00317
00318 }

```

26.59 rmol/bom/MCOptimiser.hpp File Reference

```

#include <rmol/RMOL_Types.hpp>
#include <stdair/stdair_maths_types.hpp>
#include <stdair/stdair_rm_types.hpp>

```

Classes

- class [RMOL::MCOptimiser](#)

Namespaces

- namespace [stdair](#)
- namespace [RMOL](#)

26.60 MCOptimiser.hpp

```

00001 #ifndef __RMOL_BOM_MCUTILS_HPP
00002 #define __RMOL_BOM_MCUTILS_HPP
00003
00004 // //////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////
00007 // RMOL
00008 #include <rmol/RMOL_Types.hpp>
00009 #include <stdair/stdair_maths_types.hpp>
00010 #include <stdair/stdair_rm_types.hpp>
00011
00012 // Forward declarations.

```

```

00013 namespace stdair {
00014     class LegCabin;
00015 }
00016
00017 namespace RMOL {
00019     class MCOptimiser {
00020     public:
00021
00030         static void optimalOptimisationByMCIntegration
            (stdair::LegCabin&);
00031
00035         static stdair::GeneratedDemandVector_T
00036         generateDemandVector (const stdair::MeanValue_T&,
00037                               const stdair::StdDevValue_T&, const unsigned int&);
00038
00039         static void optimisationByMCIntegration (
            stdair::LegCabin&);
00040
00041     };
00042 }
00043 #endif // __RMOL_BOM_MCUTILS_HPP

```

26.61 rmol/bom/old/DemandGeneratorList.cpp File Reference

```
#include <rmol/bom/DemandGeneratorList.hpp>
```

Namespaces

- namespace [RMOL](#)

26.62 DemandGeneratorList.cpp

```

00001 // //////////////////////////////////////
00002 // Import section
00003 // //////////////////////////////////////
00004 // RMOL
00005 #include <rmol/bom/DemandGeneratorList.hpp>
00006
00007 namespace RMOL {
00008
00009     // //////////////////////////////////////
00010     DemandGeneratorList::DemandGeneratorList
00011     () {
00012         const DistributionParameterList_T
00013         aDistributionParameterList;
00014         init (aDistributionParameterList);
00015     }
00016
00017     // //////////////////////////////////////
00018     DemandGeneratorList::
00019     DemandGeneratorList (const DemandGeneratorList
00020     & iDemandGeneratorList) {
00021         // TODO: copy the distribution parameters of the input generator list
00022         const DistributionParameterList_T
00023         aDistributionParameterList;
00024         init (aDistributionParameterList);
00025     }
00026
00027     // //////////////////////////////////////
00028     DemandGeneratorList::
00029     DemandGeneratorList (const DistributionParameterList_T
00030     & iDistributionParameterList) {
00031         init (iDistributionParameterList);
00032     }
00033
00034     // //////////////////////////////////////
00035     void DemandGeneratorList::
00036     init (const DistributionParameterList_T&
00037         iDistributionParameterList) {
00038         DistributionParameterList_T::const_iterator itParams =

```

```

00038         iDistributionParameterList.begin();
00039     for ( ; itParams != iDistributionParameterList.end(); itParams++) {
00040         const FldDistributionParameters& aParams = *itParams;
00041
00042         const Gaussian gaussianGenerator (aParams);
00043
00044         _demandGeneratorList.push_back (gaussianGenerator);
00045     }
00046 }
00047
00048 // //////////////////////////////////////
00049 void DemandGeneratorList::
00050 generateVariateList (VariateList_T& ioVariateList) const
00051 {
00052     // Iterate on the (number of) classes/buckets, n
00053     DemandGeneratorList_T::const_iterator itGenerator =
00054         _demandGeneratorList.begin();
00055     for ( ; itGenerator != _demandGeneratorList.end(); itGenerator++) {
00056         const Gaussian& gaussianGenerator = *itGenerator;
00057
00058         // Generate a random variate following the Gaussian distribution
00059         const double generatedVariate = gaussianGenerator.generateVariate ();
00060         ioVariateList.push_back (generatedVariate);
00061     }
00062 }
00063
00064 }

```

26.63 rmol/bom/old/DemandGeneratorList.hpp File Reference

```

#include <list>
#include <rmol/bom/VariateList.hpp>
#include <rmol/bom/DistributionParameterList.hpp>
#include <rmol/bom/Gaussian.hpp>

```

Classes

- class [RMOL::DemandGeneratorList](#)

Namespaces

- namespace [RMOL](#)

26.64 DemandGeneratorList.hpp

```

00001 #ifndef __RMOL_DEMANDGENERATORLIST_HPP
00002 #define __RMOL_DEMANDGENERATORLIST_HPP
00003
00004 // //////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////
00007 // STL
00008 #include <list>
00009 // RMOL
00010 #include <rmol/bom/VariateList.hpp>
00011 #include <rmol/bom/DistributionParameterList.hpp>
00012 >
00013 #include <rmol/bom/Gaussian.hpp>
00014
00015 namespace RMOL {
00016
00017     class DemandGeneratorList {
00018     protected:
00020         typedef std::list<Gaussian> DemandGeneratorList_T;
00021
00022     public:
00024         DemandGeneratorList ();
00025         DemandGeneratorList (const DemandGeneratorList
&);
00027         DemandGeneratorList (const DistributionParameterList_T
&);

```

```

00028
00030     virtual ~DemandGeneratorList();
00031
00033     void generateVariateList (VariateList_T&) const;
00034
00035 private:
00036     DemandGeneratorList_T _demandGeneratorList;
00037
00039     void init (const DistributionParameterList_T&);
00040
00041 };
00042 }
00043 #endif // __RMOL_DEMANDGENERATORLIST_HPP

```

26.65 rmol/bom/Utilities.cpp File Reference

```

#include <cassert>
#include <string>
#include <numeric>
#include <algorithm>
#include <cmath>
#include <stdair/basic/BasConst_Inventory.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/SegmentCabin.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/basic/BasConst_General.hpp>
#include <rmol/bom/Utilities.hpp>
#include <rmol/bom/GuillotineBlockHelper.hpp>

```

Namespaces

- namespace [RMOL](#)

26.66 Utilities.cpp

```

00001
00002 // //////////////////////////////////////
00003 // Import section
00004 // //////////////////////////////////////
00005 // STL
00006 #include <cassert>
00007 #include <string>
00008 #include <numeric>
00009 #include <algorithm>
00010 #include <cmath>
00011 // StdAir
00012 #include <stdair/basic/BasConst_Inventory.hpp>
00013 #include <stdair/bom/BomManager.hpp>
00014 #include <stdair/bom/SegmentCabin.hpp>
00015 #include <stdair/service/Logger.hpp>
00016 // RMOL
00017 #include <rmol/basic/BasConst_General.hpp>
00018 #include <rmol/bom/Utilities.hpp>
00019 #include <rmol/bom/GuillotineBlockHelper.hpp>
00020
00021 namespace RMOL {
00022     // //////////////////////////////////////
00023     void Utilities::
00024     computeDistributionParameters (const
00025     UnconstrainedDemandVector_T& iVector,
00026     double& ioMean, double& ioStdDev) {
00027         ioMean = 0.0; ioStdDev = 0.0;
00028         unsigned int lNbOfSamples = iVector.size();
00029         assert (lNbOfSamples > 1);
00030
00031         // Compute the mean
00032         for (UnconstrainedDemandVector_T::const_iterator itSample = iVector.begin()
00033             ;
00034             itSample != iVector.end(); ++itSample) {
00035             //STDAIR_LOG_NOTIFICATION (*itSample);
00036             ioMean += *itSample;
00037         }
00038     }
00039 }

```



```

00035     }
00036     ioMean /= lNbOfSamples;
00037
00038     // Compute the standard deviation
00039     for (UnconstrainedDemandVector_T::const_iterator itSample = iVector.begin()
;
00040         itSample != iVector.end(); ++itSample) {
00041         const double& lSample = *itSample;
00042         ioStdDev += ((lSample - ioMean) * (lSample - ioMean));
00043     }
00044     ioStdDev /= (lNbOfSamples - 1);
00045     ioStdDev = sqrt (ioStdDev);
00046
00047     // Sanity check
00048     if (ioStdDev == 0) {
00049         ioStdDev = 0.1;
00050     }
00051 }
00052
00053 // //////////////////////////////////////
00054 stdair::DCPList_T Utilities::
00055 buildRemainingDCPList (const stdair::DTD_T& iDTD) {
00056     stdair::DCPList_T oDCPList;
00057
00058     const stdair::DCPList_T lWholeDCPList = stdair::DEFAULT_DCP_LIST
;
00059     stdair::DCPList_T::const_iterator itDCP = lWholeDCPList.begin();
00060     while (itDCP != lWholeDCPList.end()) {
00061         const stdair::DCP_T& lDCP = *itDCP;
00062         if (iDTD >= lDCP) {
00063             break;
00064         }
00065         ++itDCP;
00066     }
00067     assert (itDCP != lWholeDCPList.end());
00068
00069     oDCPList.push_back (iDTD);
00070     ++itDCP;
00071     for (; itDCP != lWholeDCPList.end(); ++itDCP) {
00072         oDCPList.push_back (*itDCP);
00073     }
00074
00075     return oDCPList;
00076 }
00077
00078 // //////////////////////////////////////
00079 stdair::DCPList_T Utilities::
00080 buildRemainingDCPList2 (const stdair::DTD_T& iDTD) {
00081     stdair::DCPList_T oDCPList;
00082
00083     const stdair::DCPList_T lWholeDCPList = RMOL::DEFAULT_DCP_LIST
;
00084     stdair::DCPList_T::const_iterator itDCP = lWholeDCPList.begin();
00085     while (itDCP != lWholeDCPList.end()) {
00086         const stdair::DCP_T& lDCP = *itDCP;
00087         if (iDTD >= lDCP) {
00088             break;
00089         }
00090         ++itDCP;
00091     }
00092     assert (itDCP != lWholeDCPList.end());
00093
00094     oDCPList.push_back (iDTD);
00095     ++itDCP;
00096     for (; itDCP != lWholeDCPList.end(); ++itDCP) {
00097         oDCPList.push_back (*itDCP);
00098     }
00099
00100     return oDCPList;
00101 }
00102
00103 // //////////////////////////////////////
00104 stdair::NbOfSegments_T Utilities::
00105 getNbOfDepartedSimilarSegments (const
stdair::SegmentCabin& iSegmentCabin,
00106                                 const stdair::Date_T& iEventDate) {
00107     stdair::DTD_T lDTD = 0;
00108     // Retrieve the guillotine block.
00109     const stdair::GuillotineBlock& lGuillotineBlock =
iSegmentCabin.getGuillotineBlock();
00110     return GuillotineBlockHelper::
getNbOfSegmentAlreadyPassedThisDTD
(lGuillotineBlock, lDTD, iEventDate);
00111 }
00112
00113 }
00114
00115 }

```

26.67 rmol/bom/Utilities.hpp File Reference

```
#include <stdair/stdair_inventory_types.hpp>
#include <rmol/RMOL_Types.hpp>
```

Classes

- class [RMOL::Utilities](#)

Namespaces

- namespace [stdair](#)
- namespace [RMOL](#)

26.68 Utilities.hpp

```
00001 #ifndef __RMOL_BOM_UTILITIES_HPP
00002 #define __RMOL_BOM_UTILITIES_HPP
00003 // //////////////////////////////////////
00004 // Import section
00005 // //////////////////////////////////////
00006 // StdAir
00007 #include <stdair/stdair_inventory_types.hpp>
00008 // RMOL
00009 #include <rmol/RMOL_Types.hpp>
00010
00011 // Forward declarations
00012 namespace stdair {
00013     class SegmentCabin;
00014 }
00015
00016 namespace RMOL {
00017
00019     class Utilities {
00020     public:
00022         static void computeDistributionParameters (
00023             const UnconstrainedDemandVector_T&, double&, double&);
00027         static stdair::DCPList_T buildRemainingDCPList (const
00028             stdair::DTD_T&);
00028         static stdair::DCPList_T buildRemainingDCPList2 (
00029             const stdair::DTD_T&);
00033         static stdair::NbOfSegments_T getNbOfDepartedSimilarSegments
00034             (const stdair::SegmentCabin&, const stdair::Date_T&);
00035     };
00036
00037 }
00038
00039 #endif // __RMOL_BOM_UTILITIES_HPP
```

26.69 rmol/command/Detruncator.cpp File Reference

```
#include <cassert>
```

```

#include <stdair/basic/BasConst_Inventory.hpp>
#include <stdair/bom/GuillotineBlock.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/FlightDate.hpp>
#include <stdair/bom/SegmentDate.hpp>
#include <stdair/bom/SegmentCabin.hpp>
#include <stdair/bom/BookingClass.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/bom/GuillotineBlockHelper.hpp>
#include <rmol/bom/HistoricalBookingHolder.hpp>
#include <rmol/bom/HistoricalBooking.hpp>
#include <rmol/bom/EMDetruncator.hpp>
#include <rmol/command/Detruncator.hpp>

```

Namespaces

- namespace [RMOL](#)

26.70 Detruncator.cpp

```

00001 // //////////////////////////////////////
00002 // Import section
00003 // //////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 // StdAir
00007 #include <stdair/basic/BasConst_Inventory.hpp>
00008 #include <stdair/bom/GuillotineBlock.hpp>
00009 #include <stdair/bom/BomManager.hpp>
00010 #include <stdair/bom/FlightDate.hpp>
00011 #include <stdair/bom/SegmentDate.hpp>
00012 #include <stdair/bom/SegmentCabin.hpp>
00013 #include <stdair/bom/BookingClass.hpp>
00014 #include <stdair/service/Logger.hpp>
00015 // RMOL
00016 #include <rmol/bom/GuillotineBlockHelper.hpp>
00017 #include <rmol/bom/HistoricalBookingHolder.hpp>
00018 #include <rmol/bom/HistoricalBooking.hpp>
00019 #include <rmol/bom/EMDetruncator.hpp>
00020 #include <rmol/command/Detruncator.hpp>
00021
00022 namespace RMOL {
00023 // //////////////////////////////////////
00024 void Detruncator::
00025 unconstrainUsingAdditivePickUp (const
stdair::SegmentCabin& iSegmentCabin,
00026 BookingClassUnconstrainedDemandVectorMap_T
& ioBkgClassUncDemMap,
00027 UnconstrainedDemandVector_T&
ioQEquivalentDemandVector,
00028 const stdair::DCP_T& iDCPBegin, const stdair::DCP_T& iDCPEnd,
00029 const stdair::Date_T& iCurrentDate) {
00030
00031 // Retrieve the guillotine block.
00032 const stdair::GuillotineBlock& lGuillotineBlock =
00033 iSegmentCabin.getGuillotineBlock();
00034
00035 // Build the historical booking holders for the product-oriented bookings
00036 // of the casses and the Q-equivalent (price-oriented) bookings of the
cabin
00037 const stdair::NbOfSegments_T lNbOfUsableSegments = GuillotineBlockHelper::
00038 getNbOfSegmentAlreadyPassedThisDTD
(lGuillotineBlock, iDCPEnd,
00039 iCurrentDate);
00040
00041 // Parse the booking class list and unconstrain historical bookings.
00042 for (BookingClassUnconstrainedDemandVectorMap_T::iterator itBCUDV =
00043 ioBkgClassUncDemMap.begin(); itBCUDV != ioBkgClassUncDemMap.end();
00044 ++itBCUDV) {
00045 stdair::BookingClass* lBC_ptr = itBCUDV->first;
00046 assert (lBC_ptr != NULL);
00047 const stdair::MapKey_T& lBCKey = lBC_ptr->describeKey();
00048 const stdair::BlockIndex_T& lBlockIdx =

```

```

00049         lGuillotineBlock.getBlockIndex (lBCKey);
00050         UnconstrainedDemandVector_T& lUncDemVector =
itBCUDV->second;
00051
00052         STDAIR_LOG_DEBUG ("Unconstrain product-oriented bookings for " << lBCKey)
;
00053         // STDAIR_LOG_NOTIFICATION (lBCKey << ";" << iDCPBegin
00054         //                               << ";" << iDCPEnd);
00055         unconstrainUsingAdditivePickUp (
lGuillotineBlock, lUncDemVector,
00056                                     iDCPBegin, iDCPEnd,
00057                                     lNbOfUsableSegments, lBlockIdx);
00058     }
00059
00060     // Unconstrain the Q-equivalent bookings.
00061     // Retrieve the block index of the segment-cabin.
00062     std::ostringstream lSCMapKey;
00063     lSCMapKey << stdair::DEFAULT_SEGMENT_CABIN_VALUE_TYPE
00064               << iSegmentCabin.describeKey();
00065     const stdair::BlockIndex_T& lCabinIdx =
00066         lGuillotineBlock.getBlockIndex (lSCMapKey.str());
00067
00068     STDAIR_LOG_DEBUG ("Unconstrain price-oriented bookings");
00069     //STDAIR_LOG_NOTIFICATION (iDCPBegin << ";" << iDCPEnd);
00070     unconstrainUsingAdditivePickUp (
lGuillotineBlock, ioQEquivalentDemandVector,
00071                                     iDCPBegin, iDCPEnd, lNbOfUsableSegments,
00072                                     lCabinIdx, iSegmentCabin, iCurrentDate);
00073 }
00074
00075 // //////////////////////////////////////
00076 void Detruncator::unconstrainUsingAdditivePickUp
00077 (const stdair::GuillotineBlock& iGuillotineBlock,
00078  UnconstrainedDemandVector_T& ioUncDemVector,
00079  const stdair::DCP_T& iDCPBegin, const stdair::DCP_T& iDCPEnd,
00080  const stdair::NbOfSegments_T& iNbOfUsableSegments,
00081  const stdair::BlockIndex_T& iBlockIdx) {
00082     // TODO:
00083     stdair::NbOfSegments_T lSegBegin = 0;
00084     if (iNbOfUsableSegments > 52) lSegBegin = iNbOfUsableSegments - 52;
00085     // Retrieve the gross daily booking and availability snapshots.
00086     stdair::ConstSegmentCabinDTDRangeSnapshotView_T lBookingView =
00087         iGuillotineBlock.
getConstSegmentCabinDTDRangeProductAndPriceOrientedBookingSnapshotView (lSegBegin, iNbOfUsableSegments - 1, iDCPEnd, iDCPBegin);
00088     stdair::ConstSegmentCabinDTDRangeSnapshotView_T lAvlView =
00089         iGuillotineBlock.getConstSegmentCabinDTDRangeAvailabilitySnapshotView (
lSegBegin, iNbOfUsableSegments - 1, iDCPEnd, iDCPBegin);
00090
00091     // Browse the list of segments and build the historical booking holder.
00092     const stdair::ValueTypeIndexMap_T& lVTIdxMap =
00093         iGuillotineBlock.getValueTypeIndexMap();
00094     const unsigned int lNbOfValueTypes = lVTIdxMap.size();
00095     HistoricalBookingHolder lHBHolder;
00096     for (short i = 0; i < iNbOfUsableSegments - lSegBegin; ++i) {
00097         stdair::Flag_T lCensorshipFlag = false;
00098         stdair::NbOfBookings_T lNbOfHistoricalBkgs = 0.0;
00099         const short lNbOfDTDs = iDCPBegin - iDCPEnd + 1;
00100
00101         // Parse the DTDs during the period
00102         for (short j = 0; j < lNbOfDTDs; ++j) {
00103             // Check if the data has been censored during this day.
00104             // STDAIR_LOG_DEBUG ("i: " << i << ", NbOfValues: " << lNbOfValueTypes
00105             //                               << ", BlockIdx: " << iBlockIdx << ", j: " << j);
00106             if (lCensorshipFlag == false) {
00107                 if (lAvlView[i * lNbOfValueTypes + iBlockIdx][j] < 1.0) {
00108                     lCensorshipFlag = true;
00109                 }
00110             }
00111
00112             // Get the bookings of the day.
00113             //STDAIR_LOG_DEBUG ("Bookings of the day: " <<
lBookingView[i * lNbOfValueTypes + iBlockIdx][j]);
00114             lNbOfHistoricalBkgs += lBookingView[i * lNbOfValueTypes + iBlockIdx][j];
00115         }
00116
00117         HistoricalBooking lHistoricalBkg (lNbOfHistoricalBkgs, lCensorshipFlag);
00118         lHBHolder.addHistoricalBooking (lHistoricalBkg);
00119
00120         // DEBUG
00121         STDAIR_LOG_DEBUG ("Historical bkgs: " << lNbOfHistoricalBkgs
00122                           << ", censored: " << lCensorshipFlag);
00123         // STDAIR_LOG_NOTIFICATION (lNbOfHistoricalBkgs
00124         //                               << ";" << lCensorshipFlag);
00125     }
00126
00127     // DEBUG
00128     STDAIR_LOG_DEBUG ("Unconstrain by EM");

```

```

00129
00130     // Unconstrain the booking figures
00131     EMDetruncator::unconstrainUsingEMMethod
(lHBHolder);

00132
00133     // Add the unconstrained demand of the period to the unconstrained demand
00134     // vector.
00135     short idx = 0;
00136     for (UnconstrainedDemandVector_T::iterator itUD = ioUncDemVector.begin();
00137          itUD != ioUncDemVector.end(); ++itUD, ++idx) {
00138         *itUD += lHBHolder.getUnconstrainedDemand (idx);
00139         //STDAIR_LOG_NOTIFICATION (lHBHolder.getUnconstrainedDemand (idx));
00140     }
00141 }
00142
00143 // //////////////////////////////////////
00144 void Detruncator::unconstrainUsingAdditivePickUp
00145 (const stdair::GuillotineBlock& iGuillotineBlock,
00146  UnconstrainedDemandVector_T& ioUncDemVector,
00147  const stdair::DCP_T& iDCPBegin, const stdair::DCP_T& iDCPEnd,
00148  const stdair::NbOfSegments_T& iNbOfUsableSegments,
00149  const stdair::BlockIndex_T& iBlockIdx,
00150  const stdair::SegmentCabin& iSegmentCabin,
00151  const stdair::Date_T& iCurrentDate) {
00152     // TODO
00153     stdair::NbOfSegments_T lSegBegin = 0;
00154     if (iNbOfUsableSegments > 52) lSegBegin = iNbOfUsableSegments - 52;
00155     // Retrieve the gross daily booking and availability snapshots.
00156     stdair::ConstSegmentCabinDTDRangeSnapshotView_T lBookingView =
00157         iGuillotineBlock.
getConstSegmentCabinDTDRangeProductAndPriceOrientedBookingSnapshotView (lSegBegin, iNbOfUsableSegments - 1, iDCPEnd, iDCPBegin);
00158     stdair::ConstSegmentCabinDTDRangeSnapshotView_T lAvlView =
00159         iGuillotineBlock.getConstSegmentCabinDTDRangeAvailabilitySnapshotView (
lSegBegin, iNbOfUsableSegments - 1, iDCPEnd, iDCPBegin);

00160
00161     // Browse the list of segments and build the historical booking holder.
00162     const stdair::ValueTypeIndexMap_T& lVTIdxMap =
00163         iGuillotineBlock.getValueTypeIndexMap();
00164     const unsigned int lNbOfValueTypes = lVTIdxMap.size();
00165     HistoricalBookingHolder lHBHolder;
00166     for (short i = 0; i < iNbOfUsableSegments - lSegBegin; ++i) {
00167         stdair::Flag_T lCensorshipFlag = false;
00168         stdair::NbOfBookings_T lNbOfHistoricalBkgs = 0.0;
00169         const short lNbOfDTDs = iDCPBegin - iDCPEnd + 1;
00170
00171         // Parse the DTDs during the period
00172         for (short j = 0; j < lNbOfDTDs; ++j) {
00173             // Check if the data has been censored during this day.
00174             // STDAIR_LOG_DEBUG ("i: " << i << ", NbOfValues: " << lNbOfValueTypes
00175             //                  << ", BlockIdx: " << iBlockIdx << ", j: " << j);
00176             if (lCensorshipFlag == false) {
00177                 if (lAvlView[i * lNbOfValueTypes + iBlockIdx][j] < 1.0) {
00178                     lCensorshipFlag = true;
00179                 }
00180             }
00181
00182             // Get the bookings of the day.
00183             //STDAIR_LOG_DEBUG ("Bookings of the day: " <<
lBookingView[i * lNbOfValueTypes + iBlockIdx][j]);
00184             lNbOfHistoricalBkgs += lBookingView[i * lNbOfValueTypes + iBlockIdx][j];
00185         }
00186
00187         HistoricalBooking lHistoricalBkg (lNbOfHistoricalBkgs, lCensorshipFlag);
00188         lHBHolder.addHistoricalBooking (lHistoricalBkg);
00189
00190         // DEBUG
00191         STDAIR_LOG_DEBUG ("Historical bkgs: " << lNbOfHistoricalBkgs
00192                          << ", censored: " << lCensorshipFlag);
00193         // STDAIR_LOG_NOTIFICATION (lNbOfHistoricalBkgs
00194         //                          << "; " << lCensorshipFlag);
00195     }
00196
00197     // DEBUG
00198     STDAIR_LOG_DEBUG ("Unconstrain by EM");
00199
00200     // Unconstrain the booking figures
00201     EMDetruncator::unconstrainUsingEMMethod
(lHBHolder);

00202
00203     // Add the unconstrained demand of the period to the unconstrained demand
00204     // vector.
00205     // LOG
00206     const stdair::SegmentDate& lSegmentDate = stdair::BomManager::
getParent<stdair::SegmentDate, stdair::SegmentCabin> (iSegmentCabin);
00207     const stdair::FlightDate& lFlightDate = stdair::BomManager::
getParent<stdair::FlightDate, stdair::SegmentDate> (lSegmentDate);
00208     const stdair::Date_T& lDepDate = lFlightDate.getDepartureDate();

```

```

00211     const boost::gregorian::date_duration lDD = lDepDate - iCurrentDate;
00212     const long lDTD = lDD.days();
00213     stdair::Date_T lRefDate (2012, boost::gregorian::Jan, 01);
00214
00215     short idx = 0;
00216     for (UnconstrainedDemandVector_T::iterator itUD = ioUncDemVector.begin();
00217          itUD != ioUncDemVector.end(); ++itUD, ++idx) {
00218         *itUD += lHBHolder.getUnconstrainedDemand (idx);
00219         if (lDepDate > lRefDate) {
00220             const stdair::DateOffset_T lDateOffset (7 *(52 - idx) + 420);
00221             const stdair::Date_T lHDate = lDepDate - lDateOffset;
00222             STDAIR_LOG_NOTIFICATION (boost::gregorian::to_iso_string(lDepDate)
00223                                     << ";" << lDTD << ";" << iDCPBegin << ";"
00224                                     << iDCPEnd << ";"
00225                                     << boost::gregorian::to_iso_string (lHDate)
00226                                     << "<<lHBHolder.getUnconstrainedDemand (idx))
00227         ;
00228         STDAIR_LOG_NOTIFICATION (boost::gregorian::to_iso_string(lDepDate)
00229                                 << ";" << lDTD << ";" << iDCPBegin << ";"
00230                                 << iDCPEnd << ";"
00231                                 << boost::gregorian::to_iso_string (lHDate)
00232                                 << "<<lHBHolder.getHistoricalBooking (idx));
00233     }
00234 }
00235 }
00236
00237 // //////////////////////////////////////
00238 void Detruncator::retrieveUnconstrainedDemandForFirstDCP
00239 (const stdair::SegmentCabin& iSegmentCabin,
00240  BookingClassUnconstrainedDemandVectorMap_T
00241  & ioBkgClassUncDemVectorMap,
00242  UnconstrainedDemandVector_T&
00243  ioQEquivalentDemandVector,
00244  const stdair::DCP_T& iFirstDCP, const stdair::NbOfSegments_T& iNbOfSegments,
00245  const stdair::NbOfSegments_T& iNbOfUsedSegments){
00246
00247     // Retrieve the guillotine block.
00248     const stdair::GuillotineBlock& lGuillotineBlock =
00249         iSegmentCabin.getGuillotineBlock();
00250
00251     // Parse the booking class list and unconstrain historical bookings.
00252     for (BookingClassUnconstrainedDemandVectorMap_T::iterator itBCUDV =
00253          ioBkgClassUncDemVectorMap.begin();
00254          itBCUDV != ioBkgClassUncDemVectorMap.end(); ++itBCUDV) {
00255         stdair::BookingClass* lBC_ptr = itBCUDV->first;
00256         assert (lBC_ptr != NULL);
00257         const stdair::MapKey_T& lBCKey = lBC_ptr->describeKey();
00258         const stdair::BlockIndex_T& lBlockIdx =
00259             lGuillotineBlock.getBlockIndex (lBCKey);
00260         UnconstrainedDemandVector_T& lUncDemVector =
00261             itBCUDV->second;
00262
00263         STDAIR_LOG_DEBUG("Retrieve the unconstrained product-oriented demand for
00264 "
00265 "
00266 "
00267 "
00268 "
00269 "
00270 "
00271 "
00272 "
00273 "
00274 "
00275 "
00276 "
00277 "
00278 "
00279 "
00280 "
00281 "
00282 "
00283 "
00284 "
00285 "
00286 "
00287 "
00288 "
00289 "
00290 "
00291 "
00292 "
00293 "
00294 "
00295 "
00296 "
00297 "
00298 "
00299 "
00300 "
00301 "
00302 "
00303 "
00304 "
00305 "
00306 "
00307 "
00308 "
00309 "
00310 "
00311 "
00312 "
00313 "
00314 "
00315 "
00316 "
00317 "
00318 "
00319 "
00320 "
00321 "
00322 "
00323 "
00324 "
00325 "
00326 "
00327 "
00328 "
00329 "
00330 "
00331 "
00332 "
00333 "
00334 "
00335 "
00336 "
00337 "
00338 "
00339 "
00340 "
00341 "
00342 "
00343 "
00344 "
00345 "
00346 "
00347 "
00348 "
00349 "
00350 "
00351 "
00352 "
00353 "
00354 "
00355 "
00356 "
00357 "
00358 "
00359 "
00360 "
00361 "
00362 "
00363 "
00364 "
00365 "
00366 "
00367 "
00368 "
00369 "
00370 "
00371 "
00372 "
00373 "
00374 "
00375 "
00376 "
00377 "
00378 "
00379 "
00380 "
00381 "
00382 "
00383 "
00384 "
00385 "
00386 "
00387 "
00388 "
00389 "
00390 "
00391 "
00392 "
00393 "
00394 "
00395 "
00396 "
00397 "
00398 "
00399 "
00400 "
00401 "
00402 "
00403 "
00404 "
00405 "
00406 "
00407 "
00408 "
00409 "
00410 "
00411 "
00412 "
00413 "
00414 "
00415 "
00416 "
00417 "
00418 "
00419 "
00420 "
00421 "
00422 "
00423 "
00424 "
00425 "
00426 "
00427 "
00428 "
00429 "
00430 "
00431 "
00432 "
00433 "
00434 "
00435 "
00436 "
00437 "
00438 "
00439 "
00440 "
00441 "
00442 "
00443 "
00444 "
00445 "
00446 "
00447 "
00448 "
00449 "
00450 "
00451 "
00452 "
00453 "
00454 "
00455 "
00456 "
00457 "
00458 "
00459 "
00460 "
00461 "
00462 "
00463 "
00464 "
00465 "
00466 "
00467 "
00468 "
00469 "
00470 "
00471 "
00472 "
00473 "
00474 "
00475 "
00476 "
00477 "
00478 "
00479 "
00480 "
00481 "
00482 "
00483 "
00484 "
00485 "
00486 "
00487 "
00488 "
00489 "
00490 "
00491 "
00492 "
00493 "
00494 "
00495 "
00496 "
00497 "
00498 "
00499 "
00500 "
00501 "
00502 "
00503 "
00504 "
00505 "
00506 "
00507 "
00508 "
00509 "
00510 "
00511 "
00512 "
00513 "
00514 "
00515 "
00516 "
00517 "
00518 "
00519 "
00520 "
00521 "
00522 "
00523 "
00524 "
00525 "
00526 "
00527 "
00528 "
00529 "
00530 "
00531 "
00532 "
00533 "
00534 "
00535 "
00536 "
00537 "
00538 "
00539 "
00540 "
00541 "
00542 "
00543 "
00544 "
00545 "
00546 "
00547 "
00548 "
00549 "
00550 "
00551 "
00552 "
00553 "
00554 "
00555 "
00556 "
00557 "
00558 "
00559 "
00560 "
00561 "
00562 "
00563 "
00564 "
00565 "
00566 "
00567 "
00568 "
00569 "
00570 "
00571 "
00572 "
00573 "
00574 "
00575 "
00576 "
00577 "
00578 "
00579 "
00580 "
00581 "
00582 "
00583 "
00584 "
00585 "
00586 "
00587 "
00588 "
00589 "
00590 "
00591 "
00592 "
00593 "
00594 "
00595 "
00596 "
00597 "
00598 "
00599 "
00600 "
00601 "
00602 "
00603 "
00604 "
00605 "
00606 "
00607 "
00608 "
00609 "
00610 "
00611 "
00612 "
00613 "
00614 "
00615 "
00616 "
00617 "
00618 "
00619 "
00620 "
00621 "
00622 "
00623 "
00624 "
00625 "
00626 "
00627 "
00628 "
00629 "
00630 "
00631 "
00632 "
00633 "
00634 "
00635 "
00636 "
00637 "
00638 "
00639 "
00640 "
00641 "
00642 "
00643 "
00644 "
00645 "
00646 "
00647 "
00648 "
00649 "
00650 "
00651 "
00652 "
00653 "
00654 "
00655 "
00656 "
00657 "
00658 "
00659 "
00660 "
00661 "
00662 "
00663 "
00664 "
00665 "
00666 "
00667 "
00668 "
00669 "
00670 "
00671 "
00672 "
00673 "
00674 "
00675 "
00676 "
00677 "
00678 "
00679 "
00680 "
00681 "
00682 "
00683 "
00684 "
00685 "
00686 "
00687 "
00688 "
00689 "
00690 "
00691 "
00692 "
00693 "
00694 "
00695 "
00696 "
00697 "
00698 "
00699 "
00700 "
00701 "
00702 "
00703 "
00704 "
00705 "
00706 "
00707 "
00708 "
00709 "
00710 "
00711 "
00712 "
00713 "
00714 "
00715 "
00716 "
00717 "
00718 "
00719 "
00720 "
00721 "
00722 "
00723 "
00724 "
00725 "
00726 "
00727 "
00728 "
00729 "
00730 "
00731 "
00732 "
00733 "
00734 "
00735 "
00736 "
00737 "
00738 "
00739 "
00740 "
00741 "
00742 "
00743 "
00744 "
00745 "
00746 "
00747 "
00748 "
00749 "
00750 "
00751 "
00752 "
00753 "
00754 "
00755 "
00756 "
00757 "
00758 "
00759 "
00760 "
00761 "
00762 "
00763 "
00764 "
00765 "
00766 "
00767 "
00768 "
00769 "
00770 "
00771 "
00772 "
00773 "
00774 "
00775 "
00776 "
00777 "
00778 "
00779 "
00780 "
00781 "
00782 "
00783 "
00784 "
00785 "
00786 "
00787 "
00788 "
00789 "
00790 "
00791 "
00792 "
00793 "
00794 "
00795 "
00796 "
00797 "
00798 "
00799 "
00800 "
00801 "
00802 "
00803 "
00804 "
00805 "
00806 "
00807 "
00808 "
00809 "
00810 "
00811 "
00812 "
00813 "
00814 "
00815 "
00816 "
00817 "
00818 "
00819 "
00820 "
00821 "
00822 "
00823 "
00824 "
00825 "
00826 "
00827 "
00828 "
00829 "
00830 "
00831 "
00832 "
00833 "
00834 "
00835 "
00836 "
00837 "
00838 "
00839 "
00840 "
00841 "
00842 "
00843 "
00844 "
00845 "
00846 "
00847 "
00848 "
00849 "
00850 "
00851 "
00852 "
00853 "
00854 "
00855 "
00856 "
00857 "
00858 "
00859 "
00860 "
00861 "
00862 "
00863 "
00864 "
00865 "
00866 "
00867 "
00868 "
00869 "
00870 "
00871 "
00872 "
00873 "
00874 "
00875 "
00876 "
00877 "
00878 "
00879 "
00880 "
00881 "
00882 "
00883 "
00884 "
00885 "
00886 "
00887 "
00888 "
00889 "
00890 "
00891 "
00892 "
00893 "
00894 "
00895 "
00896 "
00897 "
00898 "
00899 "
00900 "
00901 "
00902 "
00903 "
00904 "
00905 "
00906 "
00907 "
00908 "
00909 "
00910 "
00911 "
00912 "
00913 "
00914 "
00915 "
00916 "
00917 "
00918 "
00919 "
00920 "
00921 "
00922 "
00923 "
00924 "
00925 "
00926 "
00927 "
00928 "
00929 "
00930 "
00931 "
00932 "
00933 "
00934 "
00935 "
00936 "
00937 "
00938 "
00939 "
00940 "
00941 "
00942 "
00943 "
00944 "
00945 "
00946 "
00947 "
00948 "
00949 "
00950 "
00951 "
00952 "
00953 "
00954 "
00955 "
00956 "
00957 "
00958 "
00959 "
00960 "
00961 "
00962 "
00963 "
00964 "
00965 "
00966 "
00967 "
00968 "
00969 "
00970 "
00971 "
00972 "
00973 "
00974 "
00975 "
00976 "
00977 "
00978 "
00979 "
00980 "
00981 "
00982 "
00983 "
00984 "
00985 "
00986 "
00987 "
00988 "
00989 "
00990 "
00991 "
00992 "
00993 "
00994 "
00995 "
00996 "
00997 "
00998 "
00999 "
01000 "

```

```

00290 //TODO
00291 stdair::NbOfSegments_T lSegBegin = iNbOfSegments - iNbOfUsedSegments;
00292
00293 // Retrieve the snapshots of the corresponding booking value from the
00294 // first DTD (usually 365) till the given iFirstDCP.
00295 stdair::ConstSegmentCabinDTDRangeSnapshotView_T lRangeBookingView =
00296 iGuillotineBlock.
getConstSegmentCabinDTDRangeProductAndPriceOrientedBookingSnapshotView (lSegBegin, iNbOfSegments -1, iFirstDCP, stdair:
;
00297
00298 // Sum the bookings from the first day till the given iFirstDCP in order to
00299 // get the supposing unconstrained demand for this period.
00300 const stdair::ValueTypeIndexMap_T& lVTIdxMap =
00301 iGuillotineBlock.getValueTypeIndexMap();
00302 const unsigned int lNbOfValueTypes = lVTIdxMap.size();
00303 for (int itSegment = 0; itSegment < iNbOfSegments-lSegBegin; ++itSegment) {
00304     for (int i = iFirstDCP; i <= stdair::DEFAULT_MAX_DTD; ++i) {
00305         stdair::NbOfRequests_T& lUncDemand =
00306 ioUnconstrainedDemandVector.at(itSegment);
00307         lUncDemand +=
00308 lRangeBookingView[iValueIdx + itSegment*lNbOfValueTypes][i-iFirstDCP]
;
00309     }
00310 // STDAIR_LOG_NOTIFICATION (ioUnconstrainedDemandVector.at(itSegment)
00311 // << ";" << itSegment);
00312 }
00313 }
00314
00315 // //////////////////////////////////////
00316 void Detruncator::unconstrainUsingMultiplicativePickUp
00317 (const stdair::SegmentCabin& iSegmentCabin,
00318 BookingClassUnconstrainedDemandVectorMap_T
& ioBkgClassUncDemMap,
00319 UnconstrainedDemandVector_T&
ioQEquivalentDemandVector,
00320 const stdair::DCP_T& iDCPBegin, const stdair::DCP_T& iDCPEnd,
00321 const stdair::Date_T& iCurrentDate,
00322 const stdair::NbOfSegments_T& iNbOfDepartedSegments) {
00323
00324 // Retrieve the guillotine block.
00325 const stdair::GuillotineBlock& lGuillotineBlock =
00326 iSegmentCabin.getGuillotineBlock();
00327
00328 // Build the historical booking holders for the product-oriented bookings
00329 // of the casses and the Q-equivalent (price-oriented) bookings of the
cabin
00330 const stdair::NbOfSegments_T lNbOfUsableSegments = GuillotineBlockHelper::
getNbOfSegmentAlreadyPassedThisDTD
(lGuillotineBlock, iDCPEnd,
iCurrentDate);
00332
00333
00334 // Parse the booking class list and unconstrain historical bookings.
00335 for (BookingClassUnconstrainedDemandVectorMap_T::iterator itBCUDV =
ioBkgClassUncDemMap.begin(); itBCUDV != ioBkgClassUncDemMap.end();
++itBCUDV) {
00337     stdair::BookingClass* lBC_ptr = itBCUDV->first;
00338     assert (lBC_ptr != NULL);
00339     const stdair::MapKey_T& lBCKey = lBC_ptr->describeKey();
00340     const stdair::BlockIndex_T& lBlockIdx =
lGuillotineBlock.getBlockIndex (lBCKey);
00342     UnconstrainedDemandVector_T& lUncDemVector =
itBCUDV->second;
00344
00345     STDAIR_LOG_DEBUG ("Unconstrain product-oriented bookings for " << lBCKey)
;
00346     unconstrainUsingMultiplicativePickUp (lGuillotineBlock, lUncDemVector,
iDCPBegin, iDCPEnd,
lNbOfUsableSegments, lBlockIdx,
iNbOfDepartedSegments);
00349 }
00350
00351 // Unconstrain the Q-equivalent bookings.
00352 // Retrieve the block index of the segment-cabin.
00353 std::ostream lSCMapKey;
00354 lSCMapKey << stdair::DEFAULT_SEGMENT_CABIN_VALUE_TYPE
<< iSegmentCabin.describeKey();
00356 const stdair::BlockIndex_T& lCabinIdx =
lGuillotineBlock.getBlockIndex (lSCMapKey.str());
00358
00359 STDAIR_LOG_DEBUG ("Unconstrain price-oriented bookings");
00360 unconstrainUsingMultiplicativePickUp (lGuillotineBlock,
ioQEquivalentDemandVector,
iDCPBegin, iDCPEnd,
lNbOfUsableSegments, lCabinIdx,
iNbOfDepartedSegments,
iSegmentCabin, iCurrentDate);
00366 }
00367 }

```

```

00368
00369 ///////////////////////////////////////////////////////////////////
00370 void Detruncator::unconstrainUsingMultiplicativePickUp
00371 (const stdair::GuillotineBlock& iGuillotineBlock,
00372  UnconstrainedDemandVector_T& ioUncDemVector,
00373  const stdair::DCP_T& iDCPBegin, const stdair::DCP_T& iDCPEnd,
00374  const stdair::NbOfSegments_T& iNbOfUsableSegments,
00375  const stdair::BlockIndex_T& iBlockIdx,
00376  const stdair::NbOfSegments_T& iNbOfDepartedSegments) {
00377     // TODO:
00378     stdair::NbOfSegments_T lSegBegin = 0;
00379     if (iNbOfDepartedSegments > 52) {
00380         lSegBegin = iNbOfDepartedSegments - 52;
00381     }
00382
00383     // Retrieve the gross daily booking and availability snapshots.
00384     stdair::ConstSegmentCabinDTDRangeSnapshotView_T lBookingView =
00385         iGuillotineBlock.
getConstSegmentCabinDTDRangeProductAndPriceOrientedBookingSnapshotView (lSegBegin, iNbOfUsableSegments - 1, iDCPEnd, iDCPBegin);
00386     stdair::ConstSegmentCabinDTDRangeSnapshotView_T lAvlView =
00387         iGuillotineBlock.getConstSegmentCabinDTDRangeAvailabilitySnapshotView (
lSegBegin, iNbOfUsableSegments - 1, iDCPEnd, iDCPBegin);
00388
00389     // Browse the list of segments and build the historical booking holder.
00390     const stdair::ValueTypeIndexMap_T& lVTIdxMap =
00391         iGuillotineBlock.getValueTypeIndexMap();
00392     const unsigned int lNbOfValueTypes = lVTIdxMap.size();
00393     HistoricalBookingHolder lHBHolder;
00394     std::vector<short> lDataIndexList;
00395     for (short i = 0; i < iNbOfUsableSegments - lSegBegin; ++i) {
00396         stdair::Flag_T lCensorshipFlag = false;
00397         stdair::NbOfBookings_T lNbOfHistoricalBkgs = 0.0;
00398         const short lNbOfDTDs = iDCPBegin - iDCPEnd + 1;
00399
00400         // Parse the DTDs during the period
00401         for (short j = 0; j < lNbOfDTDs; ++j) {
00402             // Check if the data has been censored during this day.
00403             // STDAIR_LOG_DEBUG ("i: " << i << ", NbOfValues: " << lNbOfValueTypes
00404             // << ", BlockIdx: " << iBlockIdx << ", j: " << j);
00405             if (lCensorshipFlag == false) {
00406                 if (lAvlView[i*lNbOfValueTypes + iBlockIdx][j] < 1.0) {
00407                     lCensorshipFlag = true;
00408                 }
00409             }
00410
00411             // Get the bookings of the day.
00412             //STDAIR_LOG_DEBUG ("Bookings of the day: " <<
lBookingView[i*lNbOfValueTypes + iBlockIdx][j]);
00413             lNbOfHistoricalBkgs += lBookingView[i*lNbOfValueTypes + iBlockIdx][j];
00414         }
00415
00416         // If there is no booking till now for this class and for this segment,
00417         // there will be no unconstraining process.
00418         stdair::NbOfRequests_T& lUncDemand = ioUncDemVector.at (i);
00419         if (lUncDemand < 1.0) {
00420             lUncDemand += lNbOfHistoricalBkgs;
00421         } else {
00422             double lBkgDemandFactor = lNbOfHistoricalBkgs / lUncDemand;
00423             HistoricalBooking lHistoricalBkg (lBkgDemandFactor, lCensorshipFlag);
00424             lHBHolder.addHistoricalBooking (lHistoricalBkg);
00425             lDataIndexList.push_back (i);
00426         }
00427
00428         // DEBUG
00429         STDAIR_LOG_DEBUG ("Historical bkgs: " << lNbOfHistoricalBkgs
<< ", censored: " << lCensorshipFlag);
00430     }
00431
00432     // DEBUG
00433     STDAIR_LOG_DEBUG ("Unconstrain by multiplicative pick-up using EM");
00434
00435     // Unconstrain the booking figures
00436     unconstrainUsingMultiplicativePickUp (
lHBHolder);
00437
00438     // Update the unconstrained demand vector.
00439     short i = 0;
00440     for (std::vector<short>::iterator itIdx = lDataIndexList.begin();
00441          itIdx != lDataIndexList.end(); ++itIdx, ++i) {
00442         short lIdx = *itIdx;
00443         stdair::NbOfRequests_T& lPastDemand = ioUncDemVector.at (lIdx);
00444         const stdair::NbOfRequests_T& lUncDemandFactorOfThisPeriod =
lHBHolder.getUnconstrainedDemand (i);
00445         lPastDemand *= (1+lUncDemandFactorOfThisPeriod);
00446     }
00447 }
00448
00449 }
00450

```



```

00451 ///////////////////////////////////////////////////////////////////
00452 void Detruncator::unconstrainUsingMultiplicativePickUp
00453 (const stdair::GuillotineBlock& iGuillotineBlock,
00454  UnconstrainedDemandVector_T& ioUncDemVector,
00455  const stdair::DCP_T& iDCPBegin, const stdair::DCP_T& iDCPEnd,
00456  const stdair::NbOfSegments_T& iNbOfUsableSegments,
00457  const stdair::BlockIndex_T& iBlockIdx,
00458  const stdair::NbOfSegments_T& iNbOfDepartedSegments,
00459  const stdair::SegmentCabin& iSegmentCabin,
00460  const stdair::Date_T& iCurrentDate) {
00461     // TODO:
00462     stdair::NbOfSegments_T lSegBegin = 0;
00463     if (iNbOfDepartedSegments > 52) {
00464         lSegBegin = iNbOfDepartedSegments - 52;
00465     }
00466
00467     // Retrieve the gross daily booking and availability snapshots.
00468     stdair::ConstSegmentCabinDTRangeSnapshotView_T lBookingView =
00469         iGuillotineBlock.
getConstSegmentCabinDTRangeProductAndPriceOrientedBookingSnapshotView (lSegBegin, iNbOfUsableSegments - 1, iDCPEnd, iDCPBegin);
00470     stdair::ConstSegmentCabinDTRangeSnapshotView_T lAvlView =
00471         iGuillotineBlock.getConstSegmentCabinDTRangeAvailabilitySnapshotView (
lSegBegin, iNbOfUsableSegments - 1, iDCPEnd, iDCPBegin);
00472
00473     // Browse the list of segments and build the historical booking holder.
00474     const stdair::ValueTypeIndexMap_T& lVTIdxMap =
00475         iGuillotineBlock.getValueTypeIndexMap();
00476     const unsigned int lNbOfValueTypes = lVTIdxMap.size();
00477     HistoricalBookingHolder lHBHolder;
00478     std::vector<short> lDataIndexList;
00479     for (short i = 0; i < iNbOfUsableSegments - lSegBegin; ++i) {
00480         stdair::Flag_T lCensorshipFlag = false;
00481         stdair::NbOfBookings_T lNbOfHistoricalBkgs = 0.0;
00482         const short lNbOfDTDs = iDCPBegin - iDCPEnd + 1;
00483
00484         // Parse the DTDs during the period
00485         for (short j = 0; j < lNbOfDTDs; ++j) {
00486             // Check if the data has been censored during this day.
00487             // STDAIR_LOG_DEBUG ("i: " << i << ", NbOfValues: " << lNbOfValueTypes
00488             //                  << ", BlockIdx: " << iBlockIdx << ", j: " << j);
00489             if (lCensorshipFlag == false) {
00490                 if (lAvlView[i * lNbOfValueTypes + iBlockIdx][j] < 1.0) {
00491                     lCensorshipFlag = true;
00492                 }
00493             }
00494
00495             // Get the bookings of the day.
00496             //STDAIR_LOG_DEBUG ("Bookings of the day: " <<
lBookingView[i * lNbOfValueTypes + iBlockIdx][j]);
00497             lNbOfHistoricalBkgs += lBookingView[i * lNbOfValueTypes + iBlockIdx][j];
00498         }
00499
00500         // If there is no booking till now for this class and for this segment,
00501         // there will be no unconstraining process.
00502         stdair::NbOfRequests_T& lUncDemand = ioUncDemVector.at (i);
00503         if (lUncDemand < 1.0) {
00504             lUncDemand += lNbOfHistoricalBkgs;
00505         } else {
00506             double lBkgDemandFactor = lNbOfHistoricalBkgs / lUncDemand;
00507             HistoricalBooking lHistoricalBkg (lBkgDemandFactor, lCensorshipFlag);
00508             lHBHolder.addHistoricalBooking (lHistoricalBkg);
00509             lDataIndexList.push_back (i);
00510         }
00511
00512         // DEBUG
00513         STDAIR_LOG_DEBUG ("Historical bkgs: " << lNbOfHistoricalBkgs
00514             << ", censored: " << lCensorshipFlag);
00515     }
00516
00517     // DEBUG
00518     STDAIR_LOG_DEBUG ("Unconstrain by multiplicative pick-up");
00519
00520     // Unconstrain the booking figures
00521     unconstrainUsingMultiplicativePickUp (
lHBHolder);
00522
00523     // Update the unconstrained demand vector.
00524     // LOG
00525     const stdair::SegmentDate& lSegmentDate = stdair::BomManager::
getParent<stdair::SegmentDate, stdair::SegmentCabin> (iSegmentCabin);
00526     const stdair::FlightDate& lFlightDate = stdair::BomManager::
getParent<stdair::FlightDate, stdair::SegmentDate> (lSegmentDate);
00527     const stdair::Date_T& lDepDate = lFlightDate.getDepartureDate();
00528     const boost::gregorian::date_duration lDD = lDepDate - iCurrentDate;
00529     const long lDTD = lDD.days();
00530     stdair::Date_T lRefDate (2012, boost::gregorian::Jan, 01);
00531
00532
00533

```

```

00534     short i = 0;
00535     for (std::vector<short>::iterator itIdx = lDataIndexList.begin();
00536          itIdx != lDataIndexList.end(); ++itIdx, ++i) {
00537         short lIdx = *itIdx;
00538         stdair::NbOfRequests_T& lPastDemand = ioUncDemVector.at (lIdx);
00539         const stdair::NbOfRequests_T& lUncDemandFactorOfThisPeriod =
00540             lHBHolder.getUnconstrainedDemand (i);
00541         const double lUncDemThisPeriod =
00542             lPastDemand * lUncDemandFactorOfThisPeriod;
00543         lPastDemand *= (1+lUncDemandFactorOfThisPeriod);
00544         if (lDepDate > lRefDate) {
00545             const stdair::DateOffset_T lDateOffset (7 *(53 - lIdx) + 420);
00546             const stdair::Date_T lHDate = lDepDate - lDateOffset;
00547             STDAIR_LOG_NOTIFICATION (boost::gregorian::to_iso_string(lDepDate)
00548                                     << ";" << lDTD << ";" << iDCPBegin << ";"
00549                                     << iDCPEnd << ";"
00550                                     << boost::gregorian::to_iso_string (lHDate)
00551                                     << ";" << lUncDemThisPeriod);
00552         }
00553     }
00554 }
00555
00556 // //////////////////////////////////////
00557 void Detruncator::
00558 unconstrainUsingMultiplicativePickUp (
00559     HistoricalBookingHolder& ioHBHolder) {
00559     // We use two loops in this algorithm. The first one is for calculating the
00560     // average of unconstrained data. The second one is fore calculating the
00561     // average of unconstrained data and the constrained data which are higher
00562     // than the first average.
00563     short lNbOfUsedData = ioHBHolder.getNbOfUncensoredData
00564 ();
00564     if (lNbOfUsedData > 0) {
00565         double lSumOfValues = 0.0;
00566         const short lNbOfData = ioHBHolder.getNbOfFlights();
00567
00568         // First loop
00569         for (short i = 0; i < lNbOfData; ++i) {
00570             if (ioHBHolder.getCensorshipFlag (i) == false) {
00571                 lSumOfValues += ioHBHolder.getHistoricalBooking (
00572 i);
00573             }
00574             double lFirstAverage = lSumOfValues / lNbOfUsedData;
00575
00576             // Second loop
00577             for (short i = 0; i < lNbOfData; ++i) {
00578                 if (ioHBHolder.getCensorshipFlag (i) == true) {
00579                     const stdair::NbOfBookings_T& lBkgs =
00580                         ioHBHolder.getHistoricalBooking (i);
00581                     if (lBkgs >= lFirstAverage) {
00582                         lSumOfValues += lBkgs;
00583                         ++lNbOfUsedData;
00584                     }
00585                 }
00586             }
00587             double lSecondAverage = lSumOfValues / lNbOfUsedData;
00588
00589             // Last loop for updating the demand.
00590             for (short i = 0; i < lNbOfData; ++i) {
00591                 if (ioHBHolder.getCensorshipFlag (i) == true) {
00592                     const stdair::NbOfBookings_T& lBkgs =
00593                         ioHBHolder.getHistoricalBooking (i);
00594                     if (lBkgs < lSecondAverage) {
00595                         ioHBHolder.setUnconstrainedDemand (
00596 lSecondAverage, i);
00597                     }
00598                 }
00599             }
00600         }
00601     }
00602 }

```

26.71 rmol/command/Detruncator.hpp File Reference

```

#include <stdair/stdair_inventory_types.hpp>
#include <rmol/RMOL_Types.hpp>

```

Classes

- class [RMOL::Detruncator](#)

Namespaces

- namespace [stdair](#)
- namespace [RMOL](#)

26.72 Detruncator.hpp

```

00001 #ifndef __RMOL_COMMAND_DETRUNCATOR_HPP
00002 #define __RMOL_COMMAND_DETRUNCATOR_HPP
00003
00004 // //////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////
00007 // StdAir
00008 #include <stdair/stdair_inventory_types.hpp>
00009 // RMOL
00010 #include <rmol/RMOL_Types.hpp>
00011
00012 // Forward declarations.
00013 namespace stdair {
00014     class GuillotineBlock;
00015     class SegmentCabin;
00016 }
00017
00018 namespace RMOL {
00019     // Forward declarations.
00020     struct HistoricalBookingHolder;
00021
00024     class Detruncator {
00025     public:
00029         static void unconstrainUsingAdditivePickUp (
00030             const stdair::SegmentCabin&,
00031             BookingClassUnconstrainedDemandVectorMap_T
00032             &,
00033             UnconstrainedDemandVector_T
00034             &,
00035             const stdair::DCP_T&, const stdair::DCP_T&,
00036             const stdair::Date_T&);
00037
00038         static void unconstrainUsingMultiplicativePickUp
00039             (const stdair::SegmentCabin&, BookingClassUnconstrainedDemandVectorMap_T
00040             &,
00041             UnconstrainedDemandVector_T&, const
00042             stdair::DCP_T&, const stdair::DCP_T&,
00043             const stdair::Date_T&, const stdair::NbOfSegments_T&);
00044
00046         static void retrieveUnconstrainedDemandForFirstDCP
00047             (const stdair::SegmentCabin&,
00048             BookingClassUnconstrainedDemandVectorMap_T
00049             &,
00050             UnconstrainedDemandVector_T&, const
00051             stdair::DCP_T&,
00052             const stdair::NbOfSegments_T&, const stdair::NbOfSegments_T&);
00053
00055         static void unconstrainUsingMultiplicativePickUp
00056             (HistoricalBookingHolder&);
00057     private:
00061         static void unconstrainUsingAdditivePickUp (
00062             const stdair::GuillotineBlock&,
00063             UnconstrainedDemandVector_T
00064             &,
00065             const stdair::DCP_T&,
00066             const stdair::DCP_T&,
00067             const stdair::NbOfSegments_T&,
00068             const stdair::BlockIndex_T&);
00070         static void unconstrainUsingAdditivePickUp (
00071             const stdair::GuillotineBlock&,
00072             UnconstrainedDemandVector_T
00073             &,
00074             const stdair::DCP_T&,
00075             const stdair::DCP_T&,
00076             const stdair::NbOfSegments_T&,
00077             const stdair::BlockIndex_T&,
00078             const stdair::SegmentCabin&,
00079             const stdair::Date_T&);

```

```

00078
00082     static void unconstrainUsingMultiplicativePickUp
00083     (const stdair::GuillotineBlock&, UnconstrainedDemandVector_T
00084     &,
00084         const stdair::DCP_T&, const stdair::DCP_T&,
00085         const stdair::NbOfSegments_T&, const stdair::BlockIndex_T&,
00086         const stdair::NbOfSegments_T&);
00087
00091     static void unconstrainUsingMultiplicativePickUp
00092     (const stdair::GuillotineBlock&, UnconstrainedDemandVector_T
00093     &,
00093         const stdair::DCP_T&, const stdair::DCP_T&,
00094         const stdair::NbOfSegments_T&, const stdair::BlockIndex_T&,
00095         const stdair::NbOfSegments_T&,
00096         const stdair::SegmentCabin&, const stdair::Date_T&);
00097
00101     static void retrieveUnconstrainedDemandForFirstDCP
00102     (const stdair::GuillotineBlock&, UnconstrainedDemandVector_T
00103     &,
00103         const stdair::DCP_T&, const stdair::BlockIndex_T&,
00104         const stdair::NbOfSegments_T&, const stdair::NbOfSegments_T&);
00105 };
00106 }
00107 #endif // __RMOL_COMMAND_DETRUNCATOR_HPP
00108
00109

```

26.73 rmol/command/Forecaster.cpp File Reference

```

#include <cassert>
#include <sstream>
#include <cmath>
#include <stdair/basic/BasConst_General.hpp>
#include <stdair/basic/BasConst_Inventory.hpp>
#include <stdair/basic/RandomGeneration.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/FlightDate.hpp>
#include <stdair/bom/LegDate.hpp>
#include <stdair/bom/SegmentDate.hpp>
#include <stdair/bom/LegCabin.hpp>
#include <stdair/bom/SegmentCabin.hpp>
#include <stdair/bom/GuillotineBlock.hpp>
#include <stdair/bom/BookingClass.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/basic/BasConst_Curves.hpp>
#include <rmol/bom/Utilities.hpp>
#include <rmol/bom/GuillotineBlockHelper.hpp>
#include <rmol/bom/HistoricalBookingHolder.hpp>
#include <rmol/bom/HistoricalBooking.hpp>
#include <rmol/bom/EMDetruncator.hpp>
#include <rmol/command/Forecaster.hpp>
#include <rmol/command/Detruncator.hpp>

```

Namespaces

- namespace [RMOL](#)

26.74 Forecaster.cpp

```

00001 // //////////////////////////////////////
00002 // Import section
00003 // //////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 #include <sstream>

```

```

00007 #include <cmath>
00008 // StdAir
00009 #include <stdair/basic/BasConst_General.hpp>
00010 #include <stdair/basic/BasConst_Inventory.hpp>
00011 #include <stdair/basic/RandomGeneration.hpp>
00012 #include <stdair/bom/BomManager.hpp>
00013 #include <stdair/bom/FlightDate.hpp>
00014 #include <stdair/bom/LegDate.hpp>
00015 #include <stdair/bom/SegmentDate.hpp>
00016 #include <stdair/bom/LegCabin.hpp>
00017 #include <stdair/bom/SegmentCabin.hpp>
00018 #include <stdair/bom/GuillotineBlock.hpp>
00019 #include <stdair/bom/BookingClass.hpp>
00020 #include <stdair/service/Logger.hpp>
00021 // RMOL
00022 #include <rmol/basic/BasConst_Curves.hpp>
00023 #include <rmol/bom/Utilities.hpp>
00024 #include <rmol/bom/GuillotineBlockHelper.hpp>
00025 #include <rmol/bom/HistoricalBookingHolder.hpp>
00026 >
00027 #include <rmol/bom/HistoricalBooking.hpp>
00028 #include <rmol/bom/EMDetruncator.hpp>
00029 #include <rmol/command/Forecaster.hpp>
00030 #include <rmol/command/Detruncator.hpp>
00031 namespace RMOL {
00032
00033 // //////////////////////////////////////
00034 bool Forecaster::
00035 forecastUsingAdditivePickUp (stdair::FlightDate&
ioFlightDate,
                                const stdair::DateTime_T& iEventTime) {
00036     // Build the offset dates.
00037     const stdair::Date_T& lEventDate = iEventTime.date();
00038     stdair::Date_T lRefDate (2012, boost::gregorian::Jan, 01);
00039
00040     //
00041     bool isSucceeded = true;
00042     const stdair::SegmentDateList_T& lSDList =
00043         stdair::BomManager::getList<stdair::SegmentDate> (ioFlightDate);
00044     for (stdair::SegmentDateList_T::const_iterator itSD = lSDList.begin();
00045          itSD != lSDList.end(); ++itSD) {
00046         stdair::SegmentDate* lSD_ptr = *itSD;
00047         assert (lSD_ptr != NULL);
00048
00049         const stdair::Date_T& lBoardingDate = lSD_ptr->getBoardingDate();
00050         const stdair::DateOffset_T lSegmentDateOffset =
00051             lBoardingDate - lEventDate;
00052         const stdair::DTD_T lSegmentDTD = lSegmentDateOffset.days();
00053
00054         // Build remaining DCP's for the segment-date.
00055         // TODO: treat the case where the segment departure is not the
00056         // same as the flight-date departure.
00057         stdair::DCPList_T lDCPList;
00058
00059         if (lEventDate < lRefDate) {
00060             lDCPList = Utilities::buildRemainingDCPList
00061 (lSegmentDTD);
00062         } else {
00063             lDCPList = Utilities::buildRemainingDCPList2
00064 (lSegmentDTD);
00065         }
00066         //
00067         const stdair::SegmentCabinList_T& lSCList =
00068             stdair::BomManager::getList<stdair::SegmentCabin> (*lSD_ptr);
00069         for (stdair::SegmentCabinList_T::const_iterator itSC = lSCList.begin();
00070              itSC != lSCList.end(); ++itSC) {
00071             stdair::SegmentCabin* lSC_ptr = *itSC;
00072             assert (lSC_ptr != NULL);
00073
00074             //
00075             // STDAIR_LOG_NOTIFICATION (ioFlightDate.getDepartureDate()
00076             // << " " << lSegmentDTD);
00077             bool isForecasted = forecastUsingAdditivePickUp
00078 (*lSC_ptr, lDCPList,
                                lEventDate);
00079             if (isForecasted == false) {
00080                 isSucceeded = false;
00081             }
00082         }
00083     }
00084     return isSucceeded;
00085 }
00086
00087 // //////////////////////////////////////

```

```

00089     bool Forecaster::
00090     forecastUsingAdditivePickUp (
00091         stdair::SegmentCabin& ioSegmentCabin,
00092         const stdair::DCPList_T& iDCPList,
00093         const stdair::Date_T& iEventDate) {
00094         // Retrieve the number of departed similar segments.
00095         stdair::NbOfSegments_T lNbOfDepartedSegments =
00096             Utilities::getNbOfDepartedSimilarSegments
00097             (ioSegmentCabin, iEventDate);
00098         // TODO
00099         if (lNbOfDepartedSegments > 52) lNbOfDepartedSegments = 52;
00100         // DEBUG
00101         STDAIR_LOG_DEBUG ("Nb of similar departed segments: "
00102             << lNbOfDepartedSegments);
00103         // If the DCP list includes only DTD0 or the number of departed
00104         // segments are less than two, remaining demand for all classes
00105         // will be set to zero.
00106         stdair::DCPList_T::const_iterator itDCP = iDCPList.begin();
00107         assert (itDCP != iDCPList.end());
00108         const stdair::DCP_T& lCurrentDTD = *itDCP;
00109         if (iDCPList.size() == 1 || lNbOfDepartedSegments < 2) {
00110             setRemainingDemandForecastToZero (ioSegmentCabin);
00111             return false;
00112         } else {
00113             // Initialise a holder for the unconstrained demand.
00114             UnconstrainedDemandVector_T
00115             lQEquivalentDemandVector (lNbOfDepartedSegments, 0.0);
00116             BookingClassUnconstrainedDemandVectorMap_T
00117             lBkgClassUncDemMap;
00118             const stdair::BookingClassList_T& lBCList =
00119                 stdair::BomManager::getList<stdair::BookingClass> (ioSegmentCabin);
00120             for (stdair::BookingClassList_T::const_iterator itBC = lBCList.begin();
00121                 itBC != lBCList.end(); ++itBC) {
00122                 stdair::BookingClass* lBC_ptr = *itBC;
00123                 assert (lBC_ptr != NULL);
00124                 std::vector<stdair::NbOfRequests_T> lUncDemandVector (
00125                     lNbOfDepartedSegments, 0.0);
00126                 bool insertionSucceeded = lBkgClassUncDemMap.insert
00127                     (BookingClassUnconstrainedDemandVectorMap_T
00128                     ::
00129                     value_type (lBC_ptr, lUncDemandVector)).second;
00130                 assert (insertionSucceeded == true);
00131             }
00132             // Build the DCP intervals and unconstrain censored booking figures for
00133             // each interval.
00134             stdair::DCPList_T::const_iterator itNextDCP = itDCP; ++itNextDCP;
00135             for (; itNextDCP != iDCPList.end(); ++itDCP, ++itNextDCP) {
00136                 const stdair::DCP_T& lCurrentDCP = *itDCP;
00137                 const stdair::DCP_T& lNextDCP = *itNextDCP;
00138                 // DEBUG
00139                 STDAIR_LOG_DEBUG ("Unconstrain demand for "
00140                     << ioSegmentCabin.describeKey()
00141                     << " and the DCP's " << lCurrentDCP << ", "
00142                     << lNextDCP);
00143                 Detruncator::unconstrainUsingAdditivePickUp
00144                 (ioSegmentCabin,
00145                     lBkgClassUncDemMap,
00146                     lQEquivalentDemandVector,
00147                     lCurrentDCP-l, lNextDCP,
00148                     iEventDate);
00149                 STDAIR_LOG_DEBUG ("Detruncation successful");
00150             }
00151             // Retrieve the FRAT5 coefficient and compute the sell-up coef.
00152             FRAT5Curve_T::const_iterator itFRAT5 =
00153                 DEFAULT_CUMULATIVE_FRAT5_CURVE.
00154                 lower_bound (lCurrentDTD);
00155             assert (itFRAT5 != DEFAULT_CUMULATIVE_FRAT5_CURVE
00156                 .end());
00157             const double lFRAT5Coef = itFRAT5->second;
00158             const double lSellUpCoef = -log(0.5) / (lFRAT5Coef - 1);
00159             forecastUsingAdditivePickUp (ioSegmentCabin,
00160                 lBkgClassUncDemMap,
00161                 lQEquivalentDemandVector, lSellUpCoef);
00162             return true;
00163         }
00164     }
00165 }
00166 // //////////////////////////////////////
00167 void Forecaster::
00168 forecastUsingAdditivePickUp (
00169     stdair::SegmentCabin& ioSegmentCabin,

```

```

00165                                     const BookingClassUnconstrainedDemandVectorMap_T
00166 & iClassUncDemMap,                                     const UnconstrainedDemandVector_T
00167 & iUncDemVector,                                     const double& iSellUpFactor) {
00168     double lPriceOriMean; double lPriceOriStdDev;
00169     Utilities::computeDistributionParameters
00170     (iUncDemVector, lPriceOriMean,
00171                                     lPriceOriStdDev);
00172     // DEBUG
00173     //STDAIR_LOG_NOTIFICATION (lPriceOriMean << " " << lPriceOriStdDev);
00174     // Retrieve the classes from low to high and compute the distributions of
00175     // product-oriented and price-oriented demand.
00176     // Retrieve the lowest class.
00177     const stdair::BookingClassList_T& lBCList =
00178         stdair::BomManager::getList<stdair::BookingClass> (ioSegmentCabin);
00179     stdair::BookingClassList_T::const_reverse_iterator itCurrentClass =
00180         lBCList.rbegin();
00181     assert (itCurrentClass != lBCList.rend());
00182     stdair::BookingClassList_T::const_reverse_iterator itNextClass =
00183         itCurrentClass;
00184     ++itNextClass;
00185     // If there is only one class in the cabin, the demand distribution of this
00186     // class is equal to the price-oriented demand distribution of the cabin.
00187     if (itNextClass == lBCList.rend()) {
00188         stdair::BookingClass* lLowestBC_ptr = *itCurrentClass;
00189         lLowestBC_ptr->setMean (lPriceOriMean);
00190         lLowestBC_ptr->setStdDev (lPriceOriStdDev);
00191     } else {
00192         // Compute the demand for higher class using the formula
00193         // Pro_sell_up_from_Q_to_F = e ^ ((y_F/y_Q - 1) * ln (0.5) / (FRAT5 - 1))
00194         for (; itNextClass != lBCList.rend(); ++itCurrentClass, ++itNextClass) {
00195             stdair::BookingClass* lCurrentBC_ptr = *itCurrentClass;
00196             assert (lCurrentBC_ptr != NULL);
00197             const stdair::Yield_T& lCurrentYield = lCurrentBC_ptr->getYield();
00198             stdair::BookingClass* lNextBC_ptr = *itNextClass;
00199             assert (lNextBC_ptr != NULL);
00200             const stdair::Yield_T& lNextYield = lNextBC_ptr->getYield();
00201             // Compute the part of price-oriented demand distributed to the
00202             // current class.
00203             const double lSellUp =
00204                 exp ((1.0 - lNextYield/lCurrentYield) * iSellUpFactor);
00205             const double lPriceOriDemMeanFrac = lPriceOriMean * (1.0 - lSellUp);
00206             const double lPriceOriDemStdDevFrac = lPriceOriStdDev * (1.0 - lSellUp);
00207         }
00208     ;
00209     // Compute the product-oriented demand distribution for the
00210     // current class.
00211     BookingClassUnconstrainedDemandVectorMap_T::const_iterator itBCUD =
00212         iClassUncDemMap.find (lCurrentBC_ptr);
00213     assert (itBCUD != iClassUncDemMap.end());
00214     const UnconstrainedDemandVector_T&
00215     lDemandVector = itBCUD->second;
00216     double lMean; double lStdDev;
00217     Utilities::computeDistributionParameters
00218     (lDemandVector, lMean, lStdDev);
00219     // Compute the demand distribution for the current class;
00220     lMean += lPriceOriDemMeanFrac;
00221     lStdDev = sqrt (lStdDev * lStdDev +
00222                     lPriceOriDemStdDevFrac * lPriceOriDemStdDevFrac);
00223     lCurrentBC_ptr->setMean (lMean);
00224     lCurrentBC_ptr->setStdDev (lStdDev);
00225     // DEBUG
00226     // STDAIR_LOG_NOTIFICATION ("Class " << lCurrentBC_ptr->describeKey()
00227     // << " , mean = " << lMean
00228     // << " , stddev = " << lStdDev);
00229     // Update the price-oriented demand
00230     lPriceOriMean *= lSellUp;
00231     lPriceOriStdDev *= lSellUp;
00232 }
00233 // Compute the demand distribution for the highest class (which is the
00234 // "current class")
00235 stdair::BookingClass* lCurrentBC_ptr = *itCurrentClass;
00236 assert (lCurrentBC_ptr != NULL);
00237 BookingClassUnconstrainedDemandVectorMap_T::const_iterator itBCUD =
00238     iClassUncDemMap.find (lCurrentBC_ptr);
00239 assert (itBCUD != iClassUncDemMap.end());
00240 const UnconstrainedDemandVector_T&
00241 lDemandVector = itBCUD->second;
00242 double lMean; double lStdDev;

```

```

00245     Utilities::computeDistributionParameters
00246     (lDemandVector, lMean, lStdDev);
00247     // Compute the demand distribution for the current class;
00248     lMean += lPriceOriMean;
00249     lStdDev = sqrt (lStdDev * lStdDev + lPriceOriStdDev * lPriceOriStdDev);
00250     lCurrentBC_ptr->setMean (lMean);
00251     lCurrentBC_ptr->setStdDev (lStdDev);
00252
00253     // DEBUG
00254     // STDAIR_LOG_NOTIFICATION ("Class " << lCurrentBC_ptr->describeKey()
00255     //                           << ", mean = " << lMean
00256     //                           << ", stddev = " << lStdDev);
00257 }
00258 }
00259
00260 // //////////////////////////////////////
00261 void Forecaster::
00262 setRemainingDemandForecastToZero (const stdair::SegmentCabin& iSegmentCabin)
00263 {
00264     // Set the demand forecast for all classes to zero.
00265     const stdair::BookingClassList_T& lBCList =
00266         stdair::BomManager::getList<stdair::BookingClass> (iSegmentCabin);
00267     for (stdair::BookingClassList_T::const_iterator itBC = lBCList.begin();
00268          itBC != lBCList.end(); ++itBC) {
00269         stdair::BookingClass* lBC_ptr = *itBC;
00270         assert (lBC_ptr != NULL);
00271         lBC_ptr->setMean (0.0);
00272     }
00273 }
00274 // //////////////////////////////////////
00275 bool Forecaster::
00276 forecastUsingMultiplicativePickUp (
00277     stdair::FlightDate& ioFlightDate,
00278     const stdair::DateTime_T& iEventTime) {
00279     // Build the offset dates.
00280     const stdair::Date_T& lEventDate = iEventTime.date();
00281
00282     //
00283     bool isSucceeded = true;
00284     const stdair::SegmentDateList_T& lSDList =
00285         stdair::BomManager::getList<stdair::SegmentDate> (ioFlightDate);
00286     for (stdair::SegmentDateList_T::const_iterator itSD = lSDList.begin();
00287          itSD != lSDList.end(); ++itSD) {
00288         stdair::SegmentDate* lSD_ptr = *itSD;
00289         assert (lSD_ptr != NULL);
00290
00291         const stdair::Date_T& lBoardingDate = lSD_ptr->getBoardingDate();
00292         const stdair::DateOffset_T lSegmentDateOffset =
00293             lBoardingDate - lEventDate;
00294         const stdair::DTD_T lSegmentDTD = lSegmentDateOffset.days();
00295
00296         //
00297         const stdair::SegmentCabinList_T& lSCList =
00298             stdair::BomManager::getList<stdair::SegmentCabin> (*lSD_ptr);
00299         for (stdair::SegmentCabinList_T::const_iterator itSC = lSCList.begin();
00300              itSC != lSCList.end(); ++itSC) {
00301             stdair::SegmentCabin* lSC_ptr = *itSC;
00302             assert (lSC_ptr != NULL);
00303
00304             bool isForecasted = forecastUsingMultiplicativePickUp
00305                 (*lSC_ptr,
00306                 lEventDate,
00307                 lSegmentDTD);
00308
00309             if (isForecasted == false) {
00310                 isSucceeded = false;
00311             }
00312         }
00313     }
00314     return isSucceeded;
00315 }
00316 // //////////////////////////////////////
00317 bool Forecaster::
00318 forecastUsingMultiplicativePickUp (
00319     stdair::SegmentCabin& ioSegmentCabin,
00320     const stdair::Date_T& iEventDate,
00321     const stdair::DTD_T& iSegmentDTD) {
00322     // Retrieving the number of anterior similar segments.
00323     const stdair::GuillotineBlock& lGuillotineBlock =
00324         ioSegmentCabin.getGuillotineBlock();
00325     stdair::NbOfSegments_T lNbOfAnteriorSimilarSegments =
00326         GuillotineBlockHelper::
00327         getNbOfSegmentAlreadyPassedThisDTD
00328         (lGuillotineBlock, iSegmentDTD,
00329         iEventDate) - 1;

```



```

00326 // Retrieve the number of departed similar segments.
00327 stdair::NbOfSegments_T lNbOfDepartedSegments =
00328 Utilities::getNbOfDepartedSimilarSegments
(ioSegmentCabin, iEventDate);
00329 // TODO:
00330 if (lNbOfDepartedSegments > 52) {
00331     lNbOfAnteriorSimilarSegments =
00332         lNbOfAnteriorSimilarSegments - lNbOfDepartedSegments + 52;
00333 }
00334 // DEBUG
00335 STDAIR_LOG_DEBUG ("Nb of anterior similar segments: "
00336 << lNbOfAnteriorSimilarSegments);
00337 // If the iSegmentDTD is the last DCP or there is no anterior similar
00338 // segment, remaining demand for all classes will be set to zero
00339 stdair::DCPList_T::const_reverse_iterator itLastDCP =
00340     stdair::DEFAULT_DCP_LIST.rbegin();
00341 assert (itLastDCP != stdair::DEFAULT_DCP_LIST.rend(
00342 ));
00343 const stdair::DCP_T& lLastDCP = *itLastDCP;
00344 if (lNbOfAnteriorSimilarSegments < 1.0 || iSegmentDTD <= lLastDCP) {
00345     setRemainingDemandForecastToZero (ioSegmentCabin);
00346     return false;
00347 } else {
00348     // Retrieve the booking figures of the first DCP and consider them
00349     // as unconstrained demand figures.
00350     stdair::DCPList_T::const_iterator itDCP = stdair::DEFAULT_DCP_LIST
00351 .begin();
00352     assert (itDCP != stdair::DEFAULT_DCP_LIST.end());
00353     const stdair::DCP_T& lFirstDCP = *itDCP;
00354     // Initialise the unconstrained demand for classes.
00355     stdair::NbOfSegments_T lNbOfUsableSegments =
00356         GuillotineBlockHelper::
00357             getNbOfSegmentAlreadyPassedThisDTD
00358 (lGuillotineBlock, lFirstDCP,
00359                                     iEventDate);
00360     // TODO
00361     unsigned short lSize = lNbOfUsableSegments;
00362     if (lNbOfDepartedSegments > 52) {
00363         lSize = lNbOfUsableSegments - lNbOfDepartedSegments + 52;
00364     }
00365     STDAIR_LOG_DEBUG ("Nb of usable similar segments: "
00366 << lNbOfUsableSegments);
00367     UnconstrainedDemandVector_T
00368 lQEquivalentDemandVector (lSize, 0.0);
00369 stdair::NbOfBookings_T lCurrentSegmentQEquivalentDemand = 0.0;
00370 BookingClassUnconstrainedDemandVectorMap_T
00371 lBkgClassUncDemVectorMap;
00372 BookingClassUnconstrainedDemandMap_T
00373 lCurrentSegmentBkgClassDemMap;
00374 const stdair::BookingClassList_T& lBCList =
00375     stdair::BomManager::getList<stdair::BookingClass> (ioSegmentCabin);
00376 for (stdair::BookingClassList_T::const_iterator itBC = lBCList.begin();
00377     itBC != lBCList.end(); ++itBC) {
00378     stdair::BookingClass* lBC_ptr = *itBC;
00379     assert (lBC_ptr != NULL);
00380     UnconstrainedDemandVector_T lUncDemandVector
00381 (lSize, 0.0);
00382     bool insertionSucceeded = lBkgClassUncDemVectorMap.
00383         insert (BookingClassUnconstrainedDemandVectorMap_T
00384 ::
00385             value_type (lBC_ptr, lUncDemandVector)).second;
00386     assert (insertionSucceeded == true);
00387     insertionSucceeded =
00388         lCurrentSegmentBkgClassDemMap.
00389         insert (BookingClassUnconstrainedDemandMap_T
00390 ::
00391             value_type (lBC_ptr, 0.0)).second;
00392     assert (insertionSucceeded == true);
00393 }
00394 Detruncator::
00395     retrieveUnconstrainedDemandForFirstDCP
00396 (ioSegmentCabin,
00397                                     lBkgClassUncDemVectorMap,
00398                                     lQEquivalentDemandVector,
00399                                     lFirstDCP, lNbOfUsableSegments,
00400                                     lSize);
00401 // Unconstrain the booking figures.
00402 stdair::DCPList_T::const_iterator itNextDCP = itDCP; ++itNextDCP;
00403 while (itNextDCP != stdair::DEFAULT_DCP_LIST.end(
00404 )) {

```

```

00401     const stdair::DCP_T& lCurrentDCP = *itDCP;
00402     const stdair::DCP_T& lNextDCP = *itNextDCP;
00403     if (lCurrentDCP <= iSegmentDTD) {
00404         break;
00405     }
00406     Detruncator::
00407     unconstrainUsingMultiplicativePickUp
00408     (ioSegmentCabin,
00409      lBkgClassUncDemVectorMap,
00410      lQEquivalentDemandVector,
00411      lCurrentDCP-1, lNextDCP,
00412      iEventDate,
00413      lNbOfDepartedSegments);
00414     ++itNextDCP; ++itDCP;
00415 }
00416 // Update the unconstrained demand for all the classes of the current
00417 // segment.
00418 lCurrentSegmentQEquivalentDemand =
00419     lQEquivalentDemandVector.at (lNbOfAnteriorSimilarSegments);
00420 BookingClassUnconstrainedDemandMap_T::iterator itBCUD =
00421     lCurrentSegmentBkgClassDemMap.begin();
00422 for (BookingClassUnconstrainedDemandVectorMap_T::iterator itBCUDV =
00423     lBkgClassUncDemVectorMap.begin();
00424     itBCUDV != lBkgClassUncDemVectorMap.end(); ++itBCUDV, ++itBCUD) {
00425     assert (itBCUD != lCurrentSegmentBkgClassDemMap.end());
00426     assert (itBCUD->first == itBCUDV->first);
00427     stdair::NbOfRequests_T& lUncDem = itBCUD->second;
00428     UnconstrainedDemandVector_T& lUncDemVector =
00429     itBCUDV->second;
00430     lUncDem = lUncDemVector.at (lNbOfAnteriorSimilarSegments);
00431 }
00432 // Forecast the remaining demand for all classes
00433 const stdair::DCP_T& lCurrentDTD = *itDCP;
00434 for (; itNextDCP != stdair::DEFAULT_DCP_LIST.end()
00435 ); ++itNextDCP, ++itDCP){
00436     const stdair::DCP_T& lCurrentDCP = *itDCP;
00437     const stdair::DCP_T& lNextDCP = *itNextDCP;
00438     forecastUsingMultiplicativePickUp (
00439     ioSegmentCabin,
00440     lBkgClassUncDemVectorMap,
00441     lQEquivalentDemandVector,
00442     lCurrentDCP-1, lNextDCP, iEventDate,
00443     lNbOfAnteriorSimilarSegments,
00444     lNbOfDepartedSegments);
00445 }
00446 // Update the remaining demand for all classes
00447 lCurrentSegmentQEquivalentDemand =
00448     lQEquivalentDemandVector.at (lNbOfAnteriorSimilarSegments)
00449     - lCurrentSegmentQEquivalentDemand;
00450 itBCUD = lCurrentSegmentBkgClassDemMap.begin();
00451 for (BookingClassUnconstrainedDemandVectorMap_T::iterator itBCUDV =
00452     lBkgClassUncDemVectorMap.begin();
00453     itBCUDV != lBkgClassUncDemVectorMap.end(); ++itBCUDV, ++itBCUD) {
00454     assert (itBCUD != lCurrentSegmentBkgClassDemMap.end());
00455     assert (itBCUD->first == itBCUDV->first);
00456     stdair::NbOfRequests_T& lUncDem = itBCUD->second;
00457     UnconstrainedDemandVector_T& lUncDemVector =
00458     itBCUDV->second;
00459     lUncDem = lUncDemVector.at (lNbOfAnteriorSimilarSegments) - lUncDem;
00460 }
00461 // Retrieve the FRAT5 coefficient and compute the sell-up coef.
00462 FRAT5Curve_T::const_iterator itFRAT5 =
00463     DEFAULT_CUMULATIVE_FRAT5_CURVE.
00464     lower_bound (lCurrentDTD);
00465     assert (itFRAT5 != DEFAULT_CUMULATIVE_FRAT5_CURVE
00466     .end());
00467     const double lFRAT5Coef = itFRAT5->second;
00468     const double lSellUpCoef = -log(0.5) / (lFRAT5Coef - 1);
00469     return forecastUsingMultiplicativePickUp
00470     (ioSegmentCabin,
00471     lCurrentSegmentBkgClassDemMap,
00472     lCurrentSegmentQEquivalentDemand
00473     ,
00474     lSellUpCoef);
00475 }
00476 }
00477 // //////////////////////////////////////
00478 void Forecaster::forecastUsingMultiplicativePickUp
00479 (const stdair::SegmentCabin& iSegmentCabin,
00480  BookingClassUnconstrainedDemandVectorMap_T

```

```

    & ioBkgClassUncDemMap,
00479     UnconstrainedDemandVector_T&
    ioQEquivalentDemandVector,
00480     const stdair::DCP_T& iDCPBegin, const stdair::DCP_T& iDCPEnd,
00481     const stdair::Date_T& iCurrentDate,
00482     const stdair::NbOfSegments_T& iNbOfAnteriorSimilarSegments,
00483     const stdair::NbOfSegments_T& iNbOfDepartedSegments) {
00484
00485     // Retrieve the guillotine block.
00486     const stdair::GuillotineBlock& lGuillotineBlock =
00487         iSegmentCabin.getGuillotineBlock();
00488
00489     // Build the historical booking holders for the product-oriented bookings
00490     // of the casses and the Q-equivalent (price-oriented) bookings of the
    cabin
00491     const stdair::NbOfSegments_T lNbOfUsableSegments = GuillotineBlockHelper::
00492         getNbOfSegmentAlreadyPassedThisDTD
    (lGuillotineBlock, iDCPEnd,
                                iCurrentDate);
00493
00494     STDAIR_LOG_DEBUG ("Nb of usable similar segments: "
00495                     << lNbOfUsableSegments);
00496
00497     if (lNbOfUsableSegments > 0) {
00498
00499         // Parse the booking class list and unconstrain historical bookings.
00500         for (BookingClassUnconstrainedDemandVectorMap_T::iterator itBCUDV =
00501             ioBkgClassUncDemMap.begin(); itBCUDV != ioBkgClassUncDemMap.end();
00502             ++itBCUDV) {
00503             stdair::BookingClass* lBC_ptr = itBCUDV->first;
00504             assert (lBC_ptr != NULL);
00505             const stdair::MapKey_T& lBCKey = lBC_ptr->describeKey();
00506             const stdair::BlockIndex_T& lBlockIdx =
00507                 lGuillotineBlock.getBlockIndex (lBCKey);
00508             UnconstrainedDemandVector_T& lUncDemVector =
00509                 itBCUDV->second;
00510
00511             STDAIR_LOG_DEBUG ("Unconstrain product-oriented bookings for "<lBCKey)
    ;
00512             forecastUsingMultiplicativePickUp (
    lGuillotineBlock, lUncDemVector,
                                iDCPBegin, iDCPEnd,
                                lNbOfUsableSegments, lBlockIdx,
                                iNbOfAnteriorSimilarSegments,
                                iNbOfDepartedSegments);
00513         }
00514
00515         // Unconstrain the Q-equivalent bookings.
00516         // Retrieve the block index of the segment-cabin.
00517         std::ostringstream lSCMapKey;
00518         lSCMapKey << stdair::DEFAULT_SEGMENT_CABIN_VALUE_TYPE
00519             << iSegmentCabin.describeKey();
00520         const stdair::BlockIndex_T& lCabinIdx =
00521             lGuillotineBlock.getBlockIndex (lSCMapKey.str());
00522
00523         STDAIR_LOG_DEBUG ("Unconstrain price-oriented bookings");
00524         forecastUsingMultiplicativePickUp (
    lGuillotineBlock,
                                ioQEquivalentDemandVector,
                                iDCPBegin, iDCPEnd,
                                lNbOfUsableSegments, lCabinIdx,
                                iNbOfAnteriorSimilarSegments,
                                iNbOfDepartedSegments,
                                iSegmentCabin, iCurrentDate);
00525     }
00526 }
00527
00528 // //////////////////////////////////////
00529 void Forecaster::forecastUsingMultiplicativePickUp
00530 (const stdair::GuillotineBlock& iGuillotineBlock,
00531  UnconstrainedDemandVector_T& ioUncDemVector,
00532  const stdair::DCP_T& iDCPBegin, const stdair::DCP_T& iDCPEnd,
00533  const stdair::NbOfSegments_T& iNbOfUsableSegments,
00534  const stdair::BlockIndex_T& iBlockIdx,
00535  const stdair::NbOfSegments_T& iNbOfAnteriorSimilarSegments,
00536  const stdair::NbOfSegments_T& iNbOfDepartedSegments) {
00537     // TODO
00538     stdair::NbOfSegments_T lSegBegin = 0;
00539     if (iNbOfDepartedSegments > 52) {
00540         lSegBegin = iNbOfDepartedSegments - 52;
00541     }
00542     // Retrieve the gross daily booking and availability snapshots.
00543     stdair::ConstSegmentCabinDTDRangeSnapshotView_T lBookingView =
00544         iGuillotineBlock.
    getConstSegmentCabinDTDRangeProductAndPriceOrientedBookingSnapshotView (lSegBegin, iNbOfUsableSegments-1, iDCPEnd, iDCPBegin)
00545     stdair::ConstSegmentCabinDTDRangeSnapshotView_T lAvlView =
00546         iGuillotineBlock.getConstSegmentCabinDTDRangeAvailabilitySnapshotView (

```

```

lSegBegin, iNbOfUsableSegments - 1, iDCPEnd, iDCPBegin);
00557
00558 // Browse the list of segments and build the historical booking holder.
00559 const stdair::ValueTypeIndexMap_T& lVTIdxMap =
00560     iGuillotineBlock.getValueTypeIndexMap();
00561 const unsigned int lNbOfValueTypes = lVTIdxMap.size();
00562 HistoricalBookingHolder lHBHolder;
00563 std::vector<short> lDataIndexList;
00564 for (short i = 0; i < iNbOfUsableSegments - lSegBegin; ++i) {
00565     stdair::Flag_T lCensorshipFlag = false;
00566     stdair::NbOfBookings_T lNbOfHistoricalBkgs = 0.0;
00567     const short lNbOfDTDs = iDCPBegin - iDCPEnd + 1;
00568
00569     // Parse the DTDs during the period
00570     for (short j = 0; j < lNbOfDTDs; ++j) {
00571         // Check if the data has been censored during this day.
00572         // STDAIR_LOG_DEBUG ("i: " << i << ", NbOfValues: " << lNbOfValueTypes
00573         // << ", BlockIdx: " << iBlockIdx << ", j: " << j);
00574         if (lCensorshipFlag == false) {
00575             if (lAvlView[i * lNbOfValueTypes + iBlockIdx][j] < 1.0) {
00576                 lCensorshipFlag = true;
00577             }
00578         }
00579
00580         // Get the bookings of the day.
00581         // STDAIR_LOG_DEBUG ("Bookings of the day: " <<
lBookingView[i * lNbOfValueTypes + iBlockIdx][j]);
00582         lNbOfHistoricalBkgs += lBookingView[i * lNbOfValueTypes + iBlockIdx][j];
00583     }
00584
00585     // If there is no booking till now for this class and for this segment,
00586     // there will be no unconstraining process.
00587     stdair::NbOfRequests_T& lUncDemand = ioUncDemVector.at (i);
00588     if (lUncDemand < 1.0) {
00589         lUncDemand += lNbOfHistoricalBkgs;
00590     } else {
00591         double lBkgDemandFactor = lNbOfHistoricalBkgs / lUncDemand;
00592         HistoricalBooking lHistoricalBkg (lBkgDemandFactor, lCensorshipFlag);
00593         lHBHolder.addHistoricalBooking (lHistoricalBkg);
00594         lDataIndexList.push_back (i);
00595     }
00596
00597     // DEBUG
00598     STDAIR_LOG_DEBUG ("Historical bkgs: " << lNbOfHistoricalBkgs
00599     << ", censored: " << lCensorshipFlag);
00600 }
00601
00602 // DEBUG
00603 STDAIR_LOG_DEBUG ("Unconstrain by multiplicative pick-up using EM");
00604
00605 // Unconstrain the booking figures
00606 Detruncator::unconstrainUsingMultiplicativePickUp
(lHBHolder);
00607
00608 // Update the unconstrained demand vector.
00609 short i = 0;
00610 for (std::vector<short>::iterator itIdx = lDataIndexList.begin();
00611     itIdx != lDataIndexList.end(); ++itIdx, ++i) {
00612     short lIdx = *itIdx;
00613     stdair::NbOfRequests_T& lPastDemand = ioUncDemVector.at (lIdx);
00614     const stdair::NbOfRequests_T& lUncDemandFactorOfThisPeriod =
00615         lHBHolder.getUnconstrainedDemand (i);
00616     lPastDemand *= (1 + lUncDemandFactorOfThisPeriod);
00617 }
00618
00619 // Update the unconstrained demand for the current segment.
00620 if (lHBHolder.getNbOfFlights() > 0) {
00621     const stdair::NbOfRequests_T& lUncDemandFactorMean =
00622         lHBHolder.getDemandMean();
00623     stdair::NbOfRequests_T& lPastDemand =
00624         ioUncDemVector.at (iNbOfAnteriorSimilarSegments);
00625     lPastDemand *= (1 + lUncDemandFactorMean);
00626 }
00627 }
00628
00629 ///////////////////////////////////////////////////////////////////
00630 void Forecaster::forecastUsingMultiplicativePickUp
00631 (const stdair::GuillotineBlock& iGuillotineBlock,
00632  UnconstrainedDemandVector_T& ioUncDemVector,
00633  const stdair::DCP_T& iDCPBegin, const stdair::DCP_T& iDCPEnd,
00634  const stdair::NbOfSegments_T& iNbOfUsableSegments,
00635  const stdair::BlockIndex_T& iBlockIdx,
00636  const stdair::NbOfSegments_T& iNbOfAnteriorSimilarSegments,
00637  const stdair::NbOfSegments_T& iNbOfDepartedSegments,
00638  const stdair::SegmentCabin& iSegmentCabin,
00639  const stdair::Date_T& iCurrentDate) {
00640     // TODO

```

```

00641     stdair::NbOfSegments_T lSegBegin = 0;
00642     if (iNbOfDepartedSegments > 52) {
00643         lSegBegin = iNbOfDepartedSegments - 52;
00644     }
00645     // Retrieve the gross daily booking and availability snapshots.
00646     stdair::ConstSegmentCabinDTDRangeSnapshotView_T lBookingView =
00647         iGuillotineBlock.
getConstSegmentCabinDTDRangeProductAndPriceOrientedBookingSnapshotView (lSegBegin, iNbOfUsableSegments -1, iDCPEnd, iDCPBegin);
00648     stdair::ConstSegmentCabinDTDRangeSnapshotView_T lAvlView =
00649         iGuillotineBlock.getConstSegmentCabinDTDRangeAvailabilitySnapshotView (
lSegBegin, iNbOfUsableSegments -1, iDCPEnd, iDCPBegin);
00650
00651     // Browse the list of segments and build the historical booking holder.
00652     const stdair::ValueTypeIndexMap_T& lVTIdxMap =
00653         iGuillotineBlock.getValueTypeIndexMap();
00654     const unsigned int lNbOfValueTypes = lVTIdxMap.size();
00655     HistoricalBookingHolder lHBHolder;
00656     std::vector<short> lDataIndexList;
00657     for (short i = 0; i < iNbOfUsableSegments-lSegBegin; ++i) {
00658         stdair::Flag_T lCensorshipFlag = false;
00659         stdair::NbOfBookings_T lNbOfHistoricalBkgs = 0.0;
00660         const short lNbOfDTDs = iDCPBegin - iDCPEnd + 1;
00661
00662         // Parse the DTDs during the period
00663         for (short j = 0; j < lNbOfDTDs; ++j) {
00664             // Check if the data has been censored during this day.
00665             // STDAIR_LOG_DEBUG ("i: " << i << ", NbOfValues: " << lNbOfValueTypes
00666             // << ", BlockIdx: " << iBlockIdx << ", j: " << j);
00667             if (lCensorshipFlag == false) {
00668                 if (lAvlView[i*lNbOfValueTypes + iBlockIdx][j] < 1.0) {
00669                     lCensorshipFlag = true;
00670                 }
00671             }
00672
00673             // Get the bookings of the day.
00674             // STDAIR_LOG_DEBUG ("Bookings of the day: " <<
lBookingView[i*lNbOfValueTypes + iBlockIdx][j]);
00675             lNbOfHistoricalBkgs += lBookingView[i*lNbOfValueTypes + iBlockIdx][j];
00676         }
00677
00678         // If there is no booking till now for this class and for this segment,
00679         // there will be no unconstraining process.
00680         stdair::NbOfRequests_T& lUncDemand = ioUncDemVector.at (i);
00681         if (lUncDemand < 1.0) {
00682             lUncDemand += lNbOfHistoricalBkgs;
00683         } else {
00684             double lBkgDemandFactor = lNbOfHistoricalBkgs / lUncDemand;
00685             HistoricalBooking lHistoricalBkg (lBkgDemandFactor, lCensorshipFlag);
00686             lHBHolder.addHistoricalBooking (lHistoricalBkg);
00687             lDataIndexList.push_back (i);
00688         }
00689
00690         // DEBUG
00691         STDAIR_LOG_DEBUG ("Historical bkgs: " << lNbOfHistoricalBkgs
00692             << ", censored: " << lCensorshipFlag);
00693     }
00694
00695     // DEBUG
00696     STDAIR_LOG_DEBUG ("Unconstrain by multiplicative pick-up using EM");
00697
00698     // Unconstrain the booking figures
00699     Detruncator::unconstrainUsingMultiplicativePickUp
(lHBHolder);
00700
00701     // Update the unconstrained demand vector.
00702     // LOG
00703     const stdair::SegmentDate& lSegmentDate = stdair::BomManager::
getParent<stdair::SegmentDate, stdair::SegmentCabin> (iSegmentCabin);
00704     const stdair::FlightDate& lFlightDate = stdair::BomManager::
getParent<stdair::FlightDate, stdair::SegmentDate> (lSegmentDate);
00705     const stdair::Date_T& lDepDate = lFlightDate.getDepartureDate();
00706     const boost::gregorian::date_duration lDD = lDepDate - iCurrentDate;
00707     const long lDTD = lDD.days();
00708     stdair::Date_T lRefDate (2012, boost::gregorian::Jan, 01);
00709     short i = 0;
00710     for (std::vector<short>::iterator itIdx = lDataIndexList.begin();
00711         itIdx != lDataIndexList.end(); ++itIdx, ++i) {
00712         short lIdx = *itIdx;
00713         stdair::NbOfRequests_T& lPastDemand = ioUncDemVector.at (lIdx);
00714         const stdair::NbOfRequests_T& lUncDemandFactorOfThisPeriod =
lHBHolder.getUnconstrainedDemand (i);
00715         const double lUncDemThisPeriod =
lPastDemand * lUncDemandFactorOfThisPeriod;
00716         const double lQEBkgThisPeriod =
lPastDemand * lHBHolder.getHistoricalBooking (i);
00717         lPastDemand *= (1+lUncDemandFactorOfThisPeriod);
00718         if (lDepDate > lRefDate) {

```

```

00724         const stdair::DateOffset_T lDateOffset (7 *(52 - i) + 420);
00725         const stdair::Date_T lHDate = lDepDate - lDateOffset;
00726         STDAIR_LOG_NOTIFICATION (boost::gregorian::to_iso_string(lDepDate)
00727             << ";" << lDTD << ";" << iDCPBegin << ";"
00728             << iDCPEnd << ";"
00729             << boost::gregorian::to_iso_string (lHDate)
00730             << ";" << lUncDemThisPeriod);
00731         STDAIR_LOG_NOTIFICATION (boost::gregorian::to_iso_string(lDepDate)
00732             << ";" << lDTD << ";" << iDCPBegin << ";"
00733             << iDCPEnd << ";"
00734             << boost::gregorian::to_iso_string (lHDate)
00735             << ";" << lQEBkgThisPeriod);
00736     }
00737 }
00738
00739 // Update the unconstrained demand for the current segment.
00740 if (lHBHolder.getNbOfFlights() > 0) {
00741     const stdair::NbOfRequests_T& lUncDemandFactorMean =
00742         lHBHolder.getDemandMean();
00743     stdair::NbOfRequests_T& lPastDemand =
00744         ioUncDemVector.at (iNbOfAnteriorSimilarSegments);
00745     lPastDemand *= (1+lUncDemandFactorMean);
00746 }
00747 }
00748
00749 ///////////////////////////////////////////////////////////////////
00750 bool Forecaster::
00751 forecastUsingMultiplicativePickUp (
00752     stdair::SegmentCabin& ioSegmentCabin,
00753     const BookingClassUnconstrainedDemandMap_T
00754     & iClassUncDemMap,
00755     const stdair::NbOfRequests_T& iUncDem,
00756     const double& iSellUpFactor) {
00757     double lPriceOriMean = iUncDem;
00758     double lPriceOriStdDev = sqrt (iUncDem);
00759     // DEBUG
00760     STDAIR_LOG_DEBUG ("Price-oriented demand: mean = " << lPriceOriMean
00761         << ", stddev = " << lPriceOriStdDev);
00762     // Retrieve the classes from low to high and compute the distributions of
00763     // product-oriented and price-oriented demand.
00764     // Retrieve the lowest class.
00765     const stdair::BookingClassList_T& lBCList =
00766         stdair::BomManager::getList<stdair::BookingClass> (ioSegmentCabin);
00767     stdair::BookingClassList_T::const_reverse_iterator itCurrentClass =
00768         lBCList.rbegin();
00769     assert (itCurrentClass != lBCList.rend());
00770     stdair::BookingClassList_T::const_reverse_iterator itNextClass =
00771         itCurrentClass;
00772     ++itNextClass;
00773     // If there is only one class in the cabin, the demand distribution of this
00774     // class is equal to the price-oriented demand distribution of the cabin.
00775     if (itNextClass == lBCList.rend()) {
00776         stdair::BookingClass* lLowestBC_ptr = *itCurrentClass;
00777         lLowestBC_ptr->setMean (lPriceOriMean);
00778         lLowestBC_ptr->setStdDev (lPriceOriStdDev);
00779         if (lPriceOriMean > 0) {
00780             return true;
00781         } else {
00782             return false;
00783         }
00784     } else {
00785         bool isSucceeded = false;
00786         // Compute the demand for higher class using the formula
00787         // Pro_sell_up_from_Q_to_F = e ^ ((y_F/y_Q - 1) * ln (0.5) / (FRAT5 - 1))
00788         for (; itNextClass != lBCList.rend(); ++itCurrentClass, ++itNextClass) {
00789             stdair::BookingClass* lCurrentBC_ptr = *itCurrentClass;
00790             assert (lCurrentBC_ptr != NULL);
00791             const stdair::Yield_T& lCurrentYield = lCurrentBC_ptr->getYield();
00792             stdair::BookingClass* lNextBC_ptr = *itNextClass;
00793             assert (lNextBC_ptr != NULL);
00794             const stdair::Yield_T& lNextYield = lNextBC_ptr->getYield();
00795
00796             // Compute the part of price-oriented demand distributed to the
00797             // current class.
00798             const double lSellUp =
00799                 exp ((1.0 - lNextYield/lCurrentYield) * iSellUpFactor);
00800             const double lPriceOriDemMeanFrac = lPriceOriMean * (1.0 - lSellUp);
00801             const double lPriceOriDemStdDevFrac = lPriceOriStdDev * (1.0 - lSellUp);
00802
00803             // Compute the product-oriented demand distribution for the
00804             // current class.
00805             BookingClassUnconstrainedDemandMap_T::const_iterator itBCUD =
00806                 iClassUncDemMap.find (lCurrentBC_ptr);
00807             assert (itBCUD != iClassUncDemMap.end());

```

```

00808         double lMean = itBCUD->second;
00809         double lStdDev = sqrt (lMean);
00810
00811         // Compute the demand distribution for the current class;
00812         lMean += lPriceOriDemMeanFrac;
00813         lStdDev = sqrt (lStdDev * lStdDev +
00814             lPriceOriDemStdDevFrac * lPriceOriDemStdDevFrac);
00815         lCurrentBC_ptr->setMean (lMean);
00816         lCurrentBC_ptr->setStdDev (lStdDev);
00817
00818         if (lMean > 0) {
00819             isSucceeded = true;
00820         }
00821
00822         // DEBUG
00823         STDAIR_LOG_DEBUG ("Class " << lCurrentBC_ptr->describeKey()
00824             << ", mean = " << lMean
00825             << ", stddev = " << lStdDev);
00826
00827         // Update the price-oriented demand
00828         lPriceOriMean *= lSellUp;
00829         lPriceOriStdDev *= lSellUp;
00830     }
00831
00832     // Compute the demand distribution for the highest class (which is the
00833     // "current class")
00834     stdair::BookingClass* lCurrentBC_ptr = *itCurrentClass;
00835     assert (lCurrentBC_ptr != NULL);
00836     BookingClassUnconstrainedDemandMap_T::const_iterator itBCUD =
00837         iClassUncDemMap.find (lCurrentBC_ptr);
00838     assert (itBCUD != iClassUncDemMap.end());
00839     double lMean = itBCUD->second;
00840     double lStdDev = sqrt (lMean);
00841
00842     // Compute the demand distribution for the current class;
00843     lMean += lPriceOriMean;
00844     lStdDev = sqrt (lStdDev * lStdDev + lPriceOriStdDev * lPriceOriStdDev);
00845     lCurrentBC_ptr->setMean (lMean);
00846     lCurrentBC_ptr->setStdDev (lStdDev);
00847
00848     if (lMean > 0) {
00849         isSucceeded = true;
00850     }
00851
00852     // DEBUG
00853     STDAIR_LOG_DEBUG ("Class " << lCurrentBC_ptr->describeKey()
00854         << ", mean = " << lMean
00855         << ", stddev = " << lStdDev);
00856     return isSucceeded;
00857 }
00858 }
00859
00860 }

```

26.75 rmol/command/Forecaster.hpp File Reference

```

#include <map>
#include <stdair/stdair_inventory_types.hpp>
#include <rmol/RMOL_Types.hpp>

```

Classes

- class [RMOL::Forecaster](#)

Namespaces

- namespace [stdair](#)
- namespace [RMOL](#)

26.76 Forecaster.hpp

```

00001 #ifndef __RMOL_COMMAND_FORECASTER_HPP

```

```

00002 #define __RMOL_COMMAND_FORECASTER_HPP
00003
00004 // //////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////
00007 // STL
00008 #include <map>
00009 // StdAir
00010 #include <stdair/stdair_inventory_types.hpp>
00011 // RMOL
00012 #include <rmol/RMOL_Types.hpp>
00013
00014 // Forward declarations
00015 namespace stdair {
00016     class FlightDate;
00017     class SegmentCabin;
00018     class GuillotineBlock;
00019 }
00020
00021 namespace RMOL {
00023     class Forecaster {
00024     public:
00028         static bool forecastUsingAdditivePickUp (
stdair::FlightDate&,
00029
00030
00031
00032
00033
00034
00035
00036
00037
00038
00039
00040
00041
00042
00043
00044
00045
00046
00047
00048
00049
00050
00051
00052
00053
00054
00055
00056
00057
00058
00059
00060
00061
00062
00063
00064
00065
00066
00067
00068
00069
00070
00071
00072
00073
00074
00075
00076
00077
00078
00079
00080
00081
00082
00083
00084
00085
00086
00087
00088
00089
00090
00091
00092
00093
00094
00095
00096
00097
00098
00099
00100
00101
00102
00103
00104
00105
00106
00107
00108
00109
00110
00111
00112
00113
00114
00115
00116
00117
00118
00119
00120
00121
00122
00123
00124
00125
00126
00127
00128
00129
00130
00131
00132
00133
00134
00135
00136
00137
00138
00139
00140
00141
00142
00143
00144
00145
00146
00147
00148
00149
00150
00151
00152
00153
00154
00155
00156
00157
00158
00159
00160
00161
00162
00163
00164
00165
00166
00167
00168
00169
00170
00171
00172
00173
00174
00175
00176
00177
00178
00179
00180
00181
00182
00183
00184
00185
00186
00187
00188
00189
00190
00191
00192
00193
00194
00195
00196
00197
00198
00199
00200
00201
00202
00203
00204
00205
00206
00207
00208
00209
00210
00211
00212
00213
00214
00215
00216
00217
00218
00219
00220
00221
00222
00223
00224
00225
00226
00227
00228
00229
00230
00231
00232
00233
00234
00235
00236
00237
00238
00239
00240
00241
00242
00243
00244
00245
00246
00247
00248
00249
00250
00251
00252
00253
00254
00255
00256
00257
00258
00259
00260
00261
00262
00263
00264
00265
00266
00267
00268
00269
00270
00271
00272
00273
00274
00275
00276
00277
00278
00279
00280
00281
00282
00283
00284
00285
00286
00287
00288
00289
00290
00291
00292
00293
00294
00295
00296
00297
00298
00299
00300
00301
00302
00303
00304
00305
00306
00307
00308
00309
00310
00311
00312
00313
00314
00315
00316
00317
00318
00319
00320
00321
00322
00323
00324
00325
00326
00327
00328
00329
00330
00331
00332
00333
00334
00335
00336
00337
00338
00339
00340
00341
00342
00343
00344
00345
00346
00347
00348
00349
00350
00351
00352
00353
00354
00355
00356
00357
00358
00359
00360
00361
00362
00363
00364
00365
00366
00367
00368
00369
00370
00371
00372
00373
00374
00375
00376
00377
00378
00379
00380
00381
00382
00383
00384
00385
00386
00387
00388
00389
00390
00391
00392
00393
00394
00395
00396
00397
00398
00399
00400
00401
00402
00403
00404
00405
00406
00407
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431
00432
00433
00434
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00
```



```

00093         ,
00094         ,
00095         ,
00096         ,
00101         static bool forecastUsingMultiplicativePickUp
00102         (stdair::SegmentCabin&,
00103         BookingClassUnconstrainedDemandMap_T&,
00104         ,
00105         ,
00109         static void setRemainingDemandForecastToZero (const stdair::SegmentCabin&);
00110     };
00111 }
00112 #endif // __RMOL_COMMAND_FORECASTER_HPP

```

26.77 rmol/command/InventoryParser.cpp File Reference

```

#include <sstream>
#include <fstream>
#include <cassert>
#include <stdair/stdair_inventory_types.hpp>
#include <stdair/stdair_maths_types.hpp>
#include <stdair/stdair_exceptions.hpp>
#include <stdair/basic/BasConst_DefaultObject.hpp>
#include <stdair/basic/BasConst_Inventory.hpp>
#include <stdair/basic/BasFileMgr.hpp>
#include <stdair/bom/BomRetriever.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/BomRoot.hpp>
#include <stdair/bom/Inventory.hpp>
#include <stdair/bom/FlightDate.hpp>
#include <stdair/bom/SegmentDate.hpp>
#include <stdair/bom/SegmentCabin.hpp>
#include <stdair/bom/LegDate.hpp>
#include <stdair/bom/LegCabin.hpp>
#include <stdair/bom/BookingClass.hpp>
#include <stdair/bom/VirtualClassStruct.hpp>
#include <stdair/factory/FacBom.hpp>
#include <stdair/factory/FacBomManager.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/command/InventoryParser.hpp>

```

Namespaces

- namespace [RMOL](#)

26.78 InventoryParser.cpp

```

00001 // //////////////////////////////////////
00002 // Import section
00003 // //////////////////////////////////////
00004 // STL
00005 #include <sstream>
00006 #include <fstream>
00007 #include <cassert>
00008 // StdAir
00009 #include <stdair/stdair_inventory_types.hpp>
00010 #include <stdair/stdair_maths_types.hpp>
00011 #include <stdair/stdair_exceptions.hpp>
00012 #include <stdair/basic/BasConst_DefaultObject.hpp>
00013 #include <stdair/basic/BasConst_Inventory.hpp>

```

```

00014 #include <stdair/basic/BasFileMgr.hpp>
00015 #include <stdair/bom/BomRetriever.hpp>
00016 #include <stdair/bom/BomManager.hpp>
00017 #include <stdair/bom/BomRoot.hpp>
00018 #include <stdair/bom/Inventory.hpp>
00019 #include <stdair/bom/FlightDate.hpp>
00020 #include <stdair/bom/SegmentDate.hpp>
00021 #include <stdair/bom/SegmentCabin.hpp>
00022 #include <stdair/bom/LegDate.hpp>
00023 #include <stdair/bom/LegCabin.hpp>
00024 #include <stdair/bom/BookingClass.hpp>
00025 #include <stdair/bom/VirtualClassStruct.hpp>
00026 #include <stdair/factory/FacBom.hpp>
00027 #include <stdair/factory/FacBomManager.hpp>
00028 #include <stdair/service/Logger.hpp>
00029 // RMOL
00030 #include <rmol/command/InventoryParser.hpp>
00031
00032 namespace RMOL {
00033
00034 // //////////////////////////////////////
00035 bool InventoryParser::
00036 parseInputFileAndBuildBom (const std::string&
00037 iInputFileName,
00038                          stdair::BomRoot& ioBomRoot) {
00039     bool hasReadBeenSuccessful = false;
00040
00041     // Check that the file path given as input corresponds to an actual file
00042     const bool doesExistAndIsReadable =
00043         stdair::BasFileMgr::doesExistAndIsReadable (iInputFileName);
00044     if (doesExistAndIsReadable == false) {
00045         std::ostringstream oMessage;
00046         oMessage << "The input file, '" << iInputFileName
00047             << "', can not be retrieved on the file-system";
00048         throw stdair::FileNotFoundException (oMessage.str());
00049     }
00050
00051     // Retrieve the (sample) leg-cabin
00052     stdair::LegCabin& lLegCabin =
00053         stdair::BomRetriever::retrieveDummyLegCabin (ioBomRoot);
00054
00055     // Retrieve the (sample) segment-cabin
00056     stdair::SegmentCabin& lSegmentCabin =
00057         stdair::BomRetriever::retrieveDummySegmentCabin (ioBomRoot);
00058
00059     // Open the input file
00060     std::ifstream inputFile (iInputFileName.c_str());
00061     if (! inputFile) {
00062         STDAIR_LOG_ERROR ("Can not open input file '" << iInputFileName << "'");
00063         throw new stdair::FileNotFoundException ("Can not open input file '"
00064             + iInputFileName + "'");
00065     }
00066
00067     char buffer[80];
00068     double dval;
00069     short i = 1;
00070     bool hasAllPParams = true;
00071     stdair::Yield_T lYield;
00072     stdair::MeanValue_T lMean;
00073     stdair::StdDevValue_T lStdDev;
00074     stdair::BookingClassKey lBCKKey (stdair::DEFAULT_CLASS_CODE);
00075
00076     while (inputFile.getline (buffer, sizeof (buffer), ';')) {
00077         std::istringstream iStringStr (buffer);
00078
00079         if (i == 1) {
00080             hasAllPParams = true;
00081         }
00082
00083         if (iStringStr >> dval) {
00084             if (i == 1) {
00085                 lYield = dval;
00086                 // std::cout << "Yield[" << i << "] = '" << dval << "'" << std::endl;
00087             } else if (i == 2) {
00088                 lMean = dval;
00089                 // std::cout << "Mean[" << i << "] = '" << dval << "'" << std::endl;
00090             } else if (i == 3) {
00091                 lStdDev = dval;
00092                 //std::cout << "stdDev[" << i << "] = '" << dval << "'" << std::endl;
00093                 i = 0;
00094             }
00095             i++;
00096         } else {
00097             hasAllPParams = false;
00098         }
00099     }

```

```

00100     }
00101
00102     if (hasAllParams && i == 1) {
00103         stdair::BookingClass& lBookingClass =
00104             stdair::FacBom<stdair::BookingClass>::instance().create (lBCKey);
00105         stdair::FacBomManager::addToList (lSegmentCabin, lBookingClass);
00106         lBookingClass.setYield (lYield);
00107         lBookingClass.setMean (lMean);
00108         lBookingClass.setStdDev (lStdDev);
00109
00110         stdair::VirtualClassStruct lVirtualClass (lBookingClass);
00111         lVirtualClass.setYield (lYield);
00112         lVirtualClass.setMean (lMean);
00113         lVirtualClass.setStdDev (lStdDev);
00114         lLegCabin.addVirtualClass (lVirtualClass);
00115     }
00116 }
00117
00118 //
00119 if (!inputFile.eof()) {
00120     STDAIR_LOG_ERROR ("Problem when reading input file '" << iInputFileName
00121                     << "'");
00122     return hasReadBeenSuccessful;
00123 }
00124
00125 //
00126 hasReadBeenSuccessful = true;
00127 return hasReadBeenSuccessful;
00128 }
00129
00130 }

```

26.79 rmol/command/InventoryParser.hpp File Reference

```

#include <string>
#include <stdair/command/CmdAbstract.hpp>

```

Classes

- class [RMOL::InventoryParser](#)
Class filling the virtual class list (representing a list of classes/buckets) from a given input inventory.

Namespaces

- namespace [stdair](#)
- namespace [RMOL](#)

26.80 InventoryParser.hpp

```

00001 #ifndef __RMOL_CMD_INVENTORYPARSER_HPP
00002 #define __RMOL_CMD_INVENTORYPARSER_HPP
00003
00004 // //////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////
00007 // STL
00008 #include <string>
00009 // StdAir
00010 #include <stdair/command/CmdAbstract.hpp>
00011
00013 namespace stdair {
00014     class BomRoot;
00015     class LegCabin;
00016     class SegmentCabin;
00017 }
00018
00019 namespace RMOL {
00020
00025     class InventoryParser : public stdair::CmdAbstract {
00026     public:
00027

```

```

00035     static bool parseInputFileAndBuildBom (const
std::string& iInputFileName,
00036                                         stdair::BomRoot&);
00037 };
00038 }
00039 #endif // __RMOL_CMD_INVENTORYPARSER_HPP

```

26.81 rmol/command/Optimiser.cpp File Reference

```

#include <cassert>
#include <sstream>
#include <stdair/basic/BasConst_General.hpp>
#include <stdair/basic/RandomGeneration.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/FlightDate.hpp>
#include <stdair/bom/LegDate.hpp>
#include <stdair/bom/SegmentDate.hpp>
#include <stdair/bom/LegCabin.hpp>
#include <stdair/bom/SegmentCabin.hpp>
#include <stdair/bom/FareFamily.hpp>
#include <stdair/bom/BookingClass.hpp>
#include <stdair/service/Logger.hpp>
#include <rmol/bom/MCOptimiser.hpp>
#include <rmol/bom/Emsr.hpp>
#include <rmol/bom/DPOptimiser.hpp>
#include <rmol/command/Optimiser.hpp>

```

Namespaces

- namespace [RMOL](#)

26.82 Optimiser.cpp

```

00001 // //////////////////////////////////////
00002 // Import section
00003 // //////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 #include <sstream>
00007 // StdAir
00008 #include <stdair/basic/BasConst_General.hpp>
00009 #include <stdair/basic/RandomGeneration.hpp>
00010 #include <stdair/bom/BomManager.hpp>
00011 #include <stdair/bom/FlightDate.hpp>
00012 #include <stdair/bom/LegDate.hpp>
00013 #include <stdair/bom/SegmentDate.hpp>
00014 #include <stdair/bom/LegCabin.hpp>
00015 #include <stdair/bom/SegmentCabin.hpp>
00016 #include <stdair/bom/FareFamily.hpp>
00017 #include <stdair/bom/BookingClass.hpp>
00018 #include <stdair/service/Logger.hpp>
00019 // RMOL
00020 #include <rmol/bom/MCOptimiser.hpp>
00021 #include <rmol/bom/Emsr.hpp>
00022 #include <rmol/bom/DPOptimiser.hpp>
00023 #include <rmol/command/Optimiser.hpp>
00024
00025 namespace RMOL {
00026
00027 // //////////////////////////////////////
00028 void Optimiser::
00029 optimalOptimisationByMCIntegration (const
int K,
00030                                     stdair::LegCabin& ioLegCabin) {
00031     // Retrieve the segment-cabin
00032     const stdair::SegmentCabinList_T lSegmentCabinList =
00033         stdair::BomManager::getList<stdair::SegmentCabin> (ioLegCabin);
00034     stdair::SegmentCabinList_T::const_iterator itSC = lSegmentCabinList.begin()

```

```

00035 ;
00036     assert (itSC != lSegmentCabinList.end());
00037     const stdair::SegmentCabin* lSegmentCabin_ptr = *itSC;
00038     assert (lSegmentCabin_ptr != NULL);
00039
00040     // Retrieve the class list.
00041     const stdair::BookingClassList_T lBookingClassList =
00042         stdair::BomManager::getList<stdair::BookingClass> (*lSegmentCabin_ptr);
00043     stdair::RandomGeneration lSeedGenerator (stdair::DEFAULT_RANDOM_SEED);
00044
00045     // Generate the demand samples for the booking classes.
00046     for (stdair::BookingClassList_T::const_iterator itBC =
00047         lBookingClassList.begin(); itBC != lBookingClassList.end(); ++itBC)
00048     {
00049         stdair::RandomSeed_T lRandomSeed =
00050             lSeedGenerator.generateUniform01 () * 1e9;
00051         stdair::BookingClass* lBookingClass_ptr = *itBC;
00052         assert (lBookingClass_ptr != NULL);
00053         lBookingClass_ptr->generateDemandSamples (K, lRandomSeed);
00054
00055         // DEBUG
00056         //STDAIR_LOG_DEBUG ("Generating " << K << " demand samples for the class
00057
00058         //
00059         << lBookingClass_ptr->describeKey());
00060     }
00061
00062     // Call the class performing the actual algorithm
00063     MCOptimiser::optimalOptimisationByMCIntegration
00064     (ioLegCabin);
00065 }
00066
00067 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00068 void Optimiser::optimalOptimisationByDP (
00069     stdair::LegCabin& ioLegCabin) {
00070     DPOptimiser::optimalOptimisationByDP (
00071         ioLegCabin);
00072 }
00073
00074 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00075 void Optimiser::heuristicOptimisationByEmsr
00076     (stdair::LegCabin& ioLegCabin) {
00077     Emsr::heuristicOptimisationByEmsr (
00078         ioLegCabin);
00079 }
00080
00081 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00082 void Optimiser::heuristicOptimisationByEmsrA
00083     (stdair::LegCabin& ioLegCabin) {
00084     Emsr::heuristicOptimisationByEmsrA (
00085         ioLegCabin);
00086 }
00087
00088 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00089 void Optimiser::heuristicOptimisationByEmsrB
00090     (stdair::LegCabin& ioLegCabin) {
00091     Emsr::heuristicOptimisationByEmsrB (
00092         ioLegCabin);
00093 }
00094
00095 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00096 void Optimiser::optimise (stdair::FlightDate& ioFlightDate
00097 ) {
00098     // Browse the leg-cabin list and build the virtual class list for
00099     // each cabin.
00100     const stdair::LegDateList_T& lLDList =
00101         stdair::BomManager::getList<stdair::LegDate> (ioFlightDate);
00102     for (stdair::LegDateList_T::const_iterator itLD = lLDList.begin();
00103         itLD != lLDList.end(); ++itLD) {
00104         stdair::LegDate* lLD_ptr = *itLD;
00105         assert (lLD_ptr != NULL);
00106
00107         //
00108         const stdair::LegCabinList_T& lLCList =
00109             stdair::BomManager::getList<stdair::LegCabin> (*lLD_ptr);
00110         for (stdair::LegCabinList_T::const_iterator itLC = lLCList.begin();
00111             itLC != lLCList.end(); ++itLC) {
00112             stdair::LegCabin* lLC_ptr = *itLC;
00113             assert (lLC_ptr != NULL);
00114
00115             // Build the virtual class list.
00116             buildVirtualClassListForLegBasedOptimisation
00117             (*lLC_ptr);
00118
00119             // Optimise using Monte-Carlo Integration method.
00120             optimalOptimisationByMCIntegration (1
00121                 0000, *lLC_ptr);
00122         }
00123     }
00124 }

```

```

00107     }
00108 }
00109
00110 // //////////////////////////////////////
00111 void Optimiser::
00112 buildVirtualClassListForLegBasedOptimisation
(stdair::LegCabin& ioLegCabin) {
00113     // The map holding all virtual classes to be created.
00114     stdair::VirtualClassMap_T lVirtualClassMap;
00115
00116     // Retrieve the segment-cabin
00117     const stdair::SegmentCabinList_T lSegmentCabinList =
00118         stdair::BomManager::getList<stdair::SegmentCabin> (ioLegCabin);
00119     stdair::SegmentCabinList_T::const_iterator itSC = lSegmentCabinList.begin()
;
00120     assert (itSC != lSegmentCabinList.end());
00121     const stdair::SegmentCabin* lSegmentCabin_ptr = *itSC;
00122     assert (lSegmentCabin_ptr != NULL);
00123
00124     // Retrieve the class list.
00125     const stdair::BookingClassList_T lBookingClassList =
00126         stdair::BomManager::getList<stdair::BookingClass> (*lSegmentCabin_ptr);
00127
00128     // Generate the demand samples for the booking classes.
00129     for (stdair::BookingClassList_T::const_iterator itBC =
00130         lBookingClassList.begin(); itBC != lBookingClassList.end(); ++itBC)
{
00131         stdair::BookingClass* lBookingClass_ptr = *itBC;
00132         assert (lBookingClass_ptr != NULL);
00133
00134         const stdair::Yield_T& lYield = lBookingClass_ptr->getYield();
00135         stdair::VirtualClassStruct lVirtualClass (*lBookingClass_ptr);
00136         lVirtualClass.setYield (lYield);
00137         lVirtualClass.setMean (lBookingClass_ptr->getMean());
00138         lVirtualClass.setStdDev (lBookingClass_ptr->getStdDev());
00139
00140         lVirtualClassMap.insert (stdair::VirtualClassMap_T::
00141             value_type (lYield, lVirtualClass));
00142     }
00143
00144     // Browse the virtual class map from high to low yield.
00145     ioLegCabin.emptyVirtualClassList();
00146     for (stdair::VirtualClassMap_T::reverse_iterator itVC =
00147         lVirtualClassMap.rbegin(); itVC != lVirtualClassMap.rend(); ++itVC)
{
00148         stdair::VirtualClassStruct& lVC = itVC->second;
00149
00150         ioLegCabin.addVirtualClass (lVC);
00151     }
00152 }
00153
00154 // //////////////////////////////////////
00155 double Optimiser::
00156 optimiseUsingOnDForecast (stdair::FlightDate&
ioFlightDate,
00157     const bool& iReduceFluctuations) {
00158     double lMaxBPVariation = 0.0;
00159     // Check if the flight date holds a list of leg dates.
00160     // If so, retrieve it and optimise the cabins.
00161     if (stdair::BomManager::hasList<stdair::LegDate> (ioFlightDate)) {
00162         STDAIR_LOG_DEBUG ("Optimisation for the flight date: "
00163             << ioFlightDate.toString());
00164         const stdair::LegDateList_T& lLDList =
00165             stdair::BomManager::getList<stdair::LegDate> (ioFlightDate);
00166         for (stdair::LegDateList_T::const_iterator itLD = lLDList.begin();
00167             itLD != lLDList.end(); ++itLD) {
00168             stdair::LegDate* lLD_ptr = *itLD;
00169             assert (lLD_ptr != NULL);
00170
00171             //
00172             const stdair::LegCabinList_T& lLCList =
00173                 stdair::BomManager::getList<stdair::LegCabin> (*lLD_ptr);
00174             for (stdair::LegCabinList_T::const_iterator itLC = lLCList.begin();
00175                 itLC != lLCList.end(); ++itLC) {
00176                 stdair::LegCabin* lLC_ptr = *itLC;
00177                 assert (lLC_ptr != NULL);
00178                 MCOptimiser::optimisationByMCIntegration
(*lLC_ptr);
00179                 const stdair::BidPrice_T& lCurrentBidPrice =
00180                     lLC_ptr->getCurrentBidPrice();
00181                 const stdair::BidPrice_T& lPreviousBidPrice =
00182                     lLC_ptr->getPreviousBidPrice();
00183                 assert (lPreviousBidPrice != 0);
00184                 const double lBPVariation =
00185                     std::abs((lCurrentBidPrice - lPreviousBidPrice)/lPreviousBidPrice);
00186                 lMaxBPVariation = std::max(lMaxBPVariation, lBPVariation);
00187             }

```

```

00188     }
00189     }
00190     return lMaxBPVariation;
00191 }
00192
00193 }

```

26.83 rmol/command/Optimiser.hpp File Reference

```
#include <rmol/RMOL_Types.hpp>
```

Classes

- class [RMOL::Optimiser](#)

Namespaces

- namespace [stdair](#)
- namespace [RMOL](#)

26.84 Optimiser.hpp

```

00001 #ifndef __RMOL_COMMAND_OPTIMISER_HPP
00002 #define __RMOL_COMMAND_OPTIMISER_HPP
00003
00004 // //////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////
00007 // RMOL
00008 #include <rmol/RMOL_Types.hpp>
00009
00010 // Forward declarations
00011 namespace stdair {
00012     class FlightDate;
00013     class LegCabin;
00014 }
00015
00016 namespace RMOL {
00017     class Optimiser {
00018     public:
00019
00020
00032         static void optimalOptimisationByMCIntegration
00033         (const int K,
00034
00035
00036
00037
00038
00039
00040
00041
00042
00043         static void optimalOptimisationByDP (
00044         stdair::LegCabin&);
00045
00046         static void heuristicOptimisationByEmsr (
00047         stdair::LegCabin&);
00048
00049         static void heuristicOptimisationByEmsrA (
00050         stdair::LegCabin&);
00051
00052         static void heuristicOptimisationByEmsrB (
00053         stdair::LegCabin&);
00054
00055         static void optimise (stdair::FlightDate&);
00056
00057         static void buildVirtualClassListForLegBasedOptimisation
00058         (stdair::LegCabin&);
00059
00060         static double optimiseUsingOnDForecast (
00061         stdair::FlightDate&,
00062
00063
00064
00065
00066
00067         const bool& iReduceFluctuations =
00068         false);
00069     };
00070 }
00071 #endif // __RMOL_COMMAND_OPTIMISER_HPP

```

26.85 rmol/config/rmol-paths.hpp File Reference

Macros

- `#define PACKAGE "rmol"`
- `#define PACKAGE_NAME "RMOL"`
- `#define PACKAGE_VERSION "0.25.3"`
- `#define PREFIXDIR "/usr"`
- `#define EXEC_PREFIX "/usr"`
- `#define BINDIR "/usr/bin"`
- `#define LIBDIR "/usr/lib"`
- `#define LIBEXECDIR "/usr/libexec"`
- `#define SBINDIR "/usr/sbin"`
- `#define SYSCONFDIR "/usr/etc"`
- `#define INCLUDEDIR "/usr/include"`
- `#define DATAROOTDIR "/usr/share"`
- `#define DATADIR "/usr/share"`
- `#define DOCDIR "/usr/share/doc/rmol-0.25.3"`
- `#define MANDIR "/usr/share/man"`
- `#define INFODIR "/usr/share/info"`
- `#define HTMLDIR "/usr/share/doc/rmol-0.25.3/html"`
- `#define PDFDIR "/usr/share/doc/rmol-0.25.3/html"`
- `#define STDAIR_SAMPLE_DIR "/usr/share/stdair/samples"`

26.85.1 Macro Definition Documentation

26.85.1.1 `#define PACKAGE "rmol"`

Definition at line 4 of file [rmol-paths.hpp](#).

26.85.1.2 `#define PACKAGE_NAME "RMOL"`

Definition at line 5 of file [rmol-paths.hpp](#).

Referenced by [readConfiguration\(\)](#).

26.85.1.3 `#define PACKAGE_VERSION "0.25.3"`

Definition at line 6 of file [rmol-paths.hpp](#).

Referenced by [readConfiguration\(\)](#).

26.85.1.4 `#define PREFIXDIR "/usr"`

Definition at line 7 of file [rmol-paths.hpp](#).

Referenced by [readConfiguration\(\)](#).

26.85.1.5 `#define EXEC_PREFIX "/usr"`

Definition at line 8 of file [rmol-paths.hpp](#).

26.85.1.6 `#define BINDIR "/usr/bin"`

Definition at line 9 of file [rmol-paths.hpp](#).

26.85.1.7 `#define LIBDIR "/usr/lib"`

Definition at line 10 of file [rmol-paths.hpp](#).

26.85.1.8 `#define LIBEXECDIR "/usr/libexec"`

Definition at line 11 of file [rmol-paths.hpp](#).

26.85.1.9 `#define SBINDIR "/usr/sbin"`

Definition at line 12 of file [rmol-paths.hpp](#).

26.85.1.10 `#define SYSCONFDIR "/usr/etc"`

Definition at line 13 of file [rmol-paths.hpp](#).

26.85.1.11 `#define INCLUDEDIR "/usr/include"`

Definition at line 14 of file [rmol-paths.hpp](#).

26.85.1.12 `#define DATAROOTDIR "/usr/share"`

Definition at line 15 of file [rmol-paths.hpp](#).

26.85.1.13 `#define DATADIR "/usr/share"`

Definition at line 16 of file [rmol-paths.hpp](#).

26.85.1.14 `#define DOCDIR "/usr/share/doc/rmol-0.25.3"`

Definition at line 17 of file [rmol-paths.hpp](#).

26.85.1.15 `#define MANDIR "/usr/share/man"`

Definition at line 18 of file [rmol-paths.hpp](#).

26.85.1.16 `#define INFODIR "/usr/share/info"`

Definition at line 19 of file [rmol-paths.hpp](#).

26.85.1.17 `#define HTMLDIR "/usr/share/doc/rmol-0.25.3/html"`

Definition at line 20 of file [rmol-paths.hpp](#).

26.85.1.18 `#define PDFDIR "/usr/share/doc/rmol-0.25.3/html"`

Definition at line 21 of file [rmol-paths.hpp](#).

26.85.1.19 `#define STDAIR_SAMPLE_DIR "/usr/share/stdair/samples"`

Definition at line 22 of file [rmol-paths.hpp](#).

26.86 rmol-paths.hpp

```
00001 #ifndef __RMOL_PATHS_HPP__
00002 #define __RMOL_PATHS_HPP__
00003
00004 #define PACKAGE "rmol"
00005 #define PACKAGE_NAME "RMOL"
00006 #define PACKAGE_VERSION "0.25.3"
00007 #define PREFIXDIR "/usr"
00008 #define EXEC_PREFIX "/usr"
00009 #define BINDIR "/usr/bin"
00010 #define LIBDIR "/usr/lib"
00011 #define LIBEXECDIR "/usr/libexec"
00012 #define SBINDIR "/usr/sbin"
00013 #define SYSCONFDIR "/usr/etc"
00014 #define INCLUDEDIR "/usr/include"
00015 #define DATAROOTDIR "/usr/share"
00016 #define DATADIR "/usr/share"
```

```

00017 #define DOCDIR "/usr/share/doc/rmol-0.25.3"
00018 #define MANDIR "/usr/share/man"
00019 #define INFODIR "/usr/share/info"
00020 #define HTMLDIR "/usr/share/doc/rmol-0.25.3/html"
00021 #define PDFDIR "/usr/share/doc/rmol-0.25.3/html"
00022 #define STDAIR_SAMPLE_DIR "/usr/share/stdair/samples"
00023
00024 #endif // __RMOL_PATHS_HPP__

```

26.87 rmol/factory/FacRmolServiceContext.cpp File Reference

```

#include <cassert>
#include <stdair/service/FacSupervisor.hpp>
#include <rmol/factory/FacRmolServiceContext.hpp>
#include <rmol/service/RMOL_ServiceContext.hpp>

```

Namespaces

- namespace [RMOL](#)

26.88 FacRmolServiceContext.cpp

```

00001 // //////////////////////////////////////
00002 // Import section
00003 // //////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 // StdAir
00007 #include <stdair/service/FacSupervisor.hpp>
00008 // RMOL
00009 #include <rmol/factory/FacRmolServiceContext.hpp>
00010 #include <rmol/service/RMOL_ServiceContext.hpp>
00011
00012 namespace RMOL {
00013
00014     FacRmolServiceContext* FacRmolServiceContext::_instance = NULL;
00015
00016     // //////////////////////////////////////
00017     FacRmolServiceContext::~FacRmolServiceContext
00018     () {
00019         _instance = NULL;
00020     }
00021
00022     // //////////////////////////////////////
00023     FacRmolServiceContext& FacRmolServiceContext::instance
00024     () {
00025         if (_instance == NULL) {
00026             _instance = new FacRmolServiceContext();
00027             assert (_instance != NULL);
00028             stdair::FacSupervisor::instance().
00029                 registerServiceFactory (_instance);
00030             return *_instance;
00031         }
00032
00033     // //////////////////////////////////////
00034     RMOL_ServiceContext& FacRmolServiceContext::create
00035     () {
00036         RMOL_ServiceContext* aServiceContext_ptr = NULL;
00037         aServiceContext_ptr = new RMOL_ServiceContext();
00038         assert (aServiceContext_ptr != NULL);
00039
00040         // The new object is added to the Bom pool
00041         _pool.push_back (aServiceContext_ptr);
00042
00043         return *aServiceContext_ptr;
00044     }
00045
00046 }

```

26.89 rmol/factory/FacRmolServiceContext.hpp File Reference

```
#include <string>
#include <stdair/stdair_basic_types.hpp>
#include <stdair/service/FacServiceAbstract.hpp>
```

Classes

- class [RMOL::FacRmolServiceContext](#)
Factory for the service context.

Namespaces

- namespace [RMOL](#)

26.90 FacRmolServiceContext.hpp

```
00001 #ifndef __RMOL_FAC_FACRMOLSERVICECONTEXT_HPP
00002 #define __RMOL_FAC_FACRMOLSERVICECONTEXT_HPP
00003
00004 // //////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////
00007 // STL
00008 #include <string>
00009 // StdAir
00010 #include <stdair/stdair_basic_types.hpp>
00011 #include <stdair/service/FacServiceAbstract.hpp>
00012
00013 namespace RMOL {
00014
00016     class RMOL_ServiceContext;
00017
00018
00022     class FacRmolServiceContext : public
stdair::FacServiceAbstract {
00023     public:
00024
00031         static FacRmolServiceContext& instance();
00032
00039         ~FacRmolServiceContext();
00040
00048         RMOL_ServiceContext& create();
00049
00050
00051     protected:
00057         FacRmolServiceContext() {}
00058
00059
00060     private:
00064         static FacRmolServiceContext* _instance;
00065     };
00066
00067 }
00068 #endif // __RMOL_FAC_FACRMOLSERVICECONTEXT_HPP
```

26.91 rmol/RMOL_Service.hpp File Reference

```
#include <string>
#include <stdair/stdair_basic_types.hpp>
#include <stdair/stdair_inventory_types.hpp>
#include <stdair/stdair_service_types.hpp>
#include <stdair/stdair_maths_types.hpp>
#include <stdair/basic/ForecastingMethod.hpp>
#include <stdair/basic/PartnershipTechnique.hpp>
#include <rmol/RMOL_Types.hpp>
```

Classes

- class [RMOL::RMOL_Service](#)
Interface for the [RMOL](#) Services.

Namespaces

- namespace [stdair](#)
- namespace [RMOL](#)

26.92 RMOL_Service.hpp

```

00001 #ifndef __RMOL_SVC_RMOL_SERVICE_HPP
00002 #define __RMOL_SVC_RMOL_SERVICE_HPP
00003
00004 // //////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////
00007 // STL
00008 #include <string>
00009 // StdAir
00010 #include <stdair/stdair_basic_types.hpp>
00011 #include <stdair/stdair_inventory_types.hpp>
00012 #include <stdair/stdair_service_types.hpp>
00013 #include <stdair/stdair_maths_types.hpp>
00014 #include <stdair/basic/ForecastingMethod.hpp>
00015 #include <stdair/basic/PartnershipTechnique.hpp>
00016 // RMOL
00017 #include <rmol/RMOL_Types.hpp>
00018
00020 namespace stdair {
00021     class FlightDate;
00022     struct BasLogParams;
00023     struct BasDBParams;
00024     class BomRoot;
00025     class AirlineClassList;
00026     class YieldFeatures;
00027     class Inventory;
00028     class OnDDate;
00029 }
00030
00031 namespace RMOL {
00032
00034     class RMOL_ServiceContext;
00035
00039     class RMOL_Service {
00040     public:
00041         // ////////////////////////////////// Constructors and destructors //////////////////////////////////
00057         RMOL_Service (const stdair::BasLogParams&, const
stdair::BasDBParams&);
00058
00070         RMOL_Service (const stdair::BasLogParams&);
00071
00087         RMOL_Service (stdair::STDAIR_ServicePtr_T);
00088
00112         void parseAndLoad (const stdair::CabinCapacity_T&
iCabinCapacity,
00113                             const stdair::Filename_T& iDemandAndClassDataFile);
00114
00118         void setUpStudyStatManager();
00119
00123         ~RMOL_Service ();
00124
00125     public:
00126         // ////////////////////////////////// Business Methods //////////////////////////////////
00133         void buildSampleBom();
00134
00138         void optimalOptimisationByMCIntegration (
const int K);
00139
00143         void optimalOptimisationByDP ();
00144
00148         void heuristicOptimisationByEmsr ();
00149
00153         void heuristicOptimisationByEmsrA ();
00154
00158         void heuristicOptimisationByEmsrB ();
00159

```

```

00163     bool optimise (stdair::FlightDate&, const stdair::DateTime_T&,
00164                   const stdair::ForecastingMethod&, const
stdair::PartnershipTechnique&);
00165
00170     // O&D based forecast
00171     void forecastOnD (const stdair::DateTime_T&);
00172
00173     stdair::YieldFeatures* getYieldFeatures(const
stdair::OnDDate&, const stdair::CabinCode_T&,
stdair::BomRoot&);
00174
00175     void forecastOnD (const stdair::YieldFeatures&, stdair::OnDDate&
,
const stdair::CabinCode_T&, const stdair::DTD_T&,
stdair::BomRoot&);
00178
00179     void setOnDForecast (const stdair::AirlineClassList&, const
stdair::MeanValue_T&,
const stdair::StdDevValue_T&, stdair::OnDDate&, const
stdair::CabinCode_T&,
stdair::BomRoot&);
00182
00183     // Single segment O&D
00184     void setOnDForecast (const stdair::AirlineCode_T&, const
stdair::Date_T&, const stdair::AirportCode_T&,
const stdair::AirportCode_T&, const
stdair::CabinCode_T&, const stdair::ClassCode_T&,
const stdair::MeanValue_T&, const
stdair::StdDevValue_T&, const stdair::Yield_T&, stdair::BomRoot&);
00188
00189     // Multiple segment O&D
00190     void setOnDForecast (const stdair::AirlineCodeList_T&, const
stdair::AirlineCode_T&,const stdair::Date_T&,
const stdair::AirportCode_T&, const
stdair::AirportCode_T&, const stdair::CabinCode_T&,
const stdair::ClassCodeList_T&, const
stdair::MeanValue_T&, const stdair::StdDevValue_T&,
const stdair::Yield_T&, stdair::BomRoot&);
00193
00194     // Initialise (or re-initialise) the demand projections in all leg cabins
00195     void resetDemandInformation (const stdair::DateTime_T
&);
00196
00197     void resetDemandInformation (const stdair::DateTime_T
&, const stdair::Inventory&);
00198
00199     /* Projection of demand */
00200
00201     // Aggregated demand at booking class level.
00202     void projectAggregatedDemandOnLegCabins(
const stdair::DateTime_T&);
00203
00204     // Static rule prorated yield
00205     void projectOnDDemandOnLegCabinsUsingYP(
const stdair::DateTime_T&);
00206
00207     // Displacement-adjusted yield
00208     void projectOnDDemandOnLegCabinsUsingDA(
const stdair::DateTime_T&);
00209
00210     // Dynamic yield proration (PF = BP_i/BP_{total}, where BP_{total} =
sum(BP_i))
00211     void projectOnDDemandOnLegCabinsUsingDYP
(const stdair::DateTime_T&);
00212
00213     void projectOnDDemandOnLegCabinsUsingDYP
(const stdair::DateTime_T&, const stdair::Inventory&);
00214
00215     // O&D-based optimisation (using demand aggregation or demand aggregation).
00216     void optimiseOnD (const stdair::DateTime_T&);
00217
00218     // O&D-based optimisation using displacement-adjusted yield.
00219     void optimiseOnDUsingRMCooperation (const
stdair::DateTime_T&);
00220
00221     // Advanced version of O&D-based optimisation using displacement-adjusted
yield.
00222     // Network optimisation instead of separate inventory optimisation.
00223     void optimiseOnDUsingAdvancedRMCooperation
(const stdair::DateTime_T&);
00224
00225     // Update bid price and send to partners
00226     void updateBidPrice (const stdair::DateTime_T&);
00227
00228     void updateBidPrice (const stdair::FlightDate&,
stdair::BomRoot&);
00229
00230     public:
00231

```

```

00232 // ////////////////////////////////// Export support methods //////////////////////////////////
00243 std::string jsonExport (const stdair::AirlineCode_T&,
00244                        const stdair::FlightNumber_T&,
00245                        const stdair::Date_T& iDepartureDate) const;
00246
00247
00248 public:
00249 // ////////////////////////////////// Display support methods //////////////////////////////////
00257 std::string csvDisplay() const;
00258
00259
00260 private:
00261 // ////////////////////////////////// Construction and Destruction helper methods //////////////////////////////////
00265 RMOL_Service();
00266
00270 RMOL_Service (const RMOL_Service&);
00271
00281 stdair::STDAIR_ServicePtr_T initStdAirService (const stdair::BasLogParams&,
00282                                                const stdair::BasDBParams&);
00283
00292 stdair::STDAIR_ServicePtr_T initStdAirService (const stdair::BasLogParams&)
;
00293
00302 void addStdAirService (stdair::STDAIR_ServicePtr_T,
00303                       const bool iOwnStdairService);
00304
00309 void initServiceContext();
00310
00317 void initRmolService();
00318
00322 void finalise();
00323
00324
00325 private:
00326 // ////////////////////////////////// Service Context //////////////////////////////////
00330 RMOL_ServiceContext* _rmolServiceContext;
00331
00333 stdair::Date_T _previousForecastDate;
00334 };
00335 }
00336 #endif // __RMOL_SVC_RMOL_SERVICE_HPP

```

26.93 rmol/RMOL_Types.hpp File Reference

```

#include <map>
#include <vector>
#include <boost/shared_ptr.hpp>
#include <stdair/stdair_inventory_types.hpp>
#include <stdair/stdair_exceptions.hpp>

```

Classes

- class [RMOL::OverbookingException](#)
Overbooking-related exception.
- class [RMOL::UnconstrainingException](#)
Unconstraining-related exception.
- class [RMOL::ForecastException](#)
Forecast-related exception.
- class [RMOL::OptimisationException](#)
Optimisation-related exception.

Namespaces

- namespace [stdair](#)
- namespace [RMOL](#)

Typedefs

- typedef boost::shared_ptr
 < RMOL_Service > RMOL::RMOL_ServicePtr_T
- typedef std::vector
 < stdair::NbOfRequests_T > RMOL::UnconstrainedDemandVector_T
- typedef std::vector
 < stdair::NbOfBookings_T > RMOL::BookingVector_T
- typedef std::vector
 < stdair::Flag_T > RMOL::FlagVector_T
- typedef std::map
 < stdair::BookingClass
 *, UnconstrainedDemandVector_T > RMOL::BookingClassUnconstrainedDemandVectorMap_T
- typedef std::map
 < stdair::BookingClass
 *, stdair::NbOfRequests_T > RMOL::BookingClassUnconstrainedDemandMap_T
- typedef std::map< const
 stdair::DTD_T, double > RMOL::FRAT5Curve_T

26.94 RMOL_Types.hpp

```

00001 #ifndef __RMOL_RMOL_TYPES_HPP
00002 #define __RMOL_RMOL_TYPES_HPP
00003
00004 // //////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////
00007 // STL
00008 #include <map>
00009 #include <vector>
00010 // Boost
00011 #include <boost/shared_ptr.hpp>
00012 // StdAir
00013 #include <stdair/stdair_inventory_types.hpp>
00014 #include <stdair/stdair_exceptions.hpp>
00015
00016 // Forward declarations.
00017 namespace stdair {
00018     class BookingClass;
00019 }
00020
00021
00022 namespace RMOL {
00023
00024     // Forward declarations
00025     class RMOL_Service;
00026
00027     // /////////// Exceptions ///////////
00031     class OverbookingException : public stdair::RootException
00032     {
00033     public:
00034         OverbookingException (const std::string& iWhat)
00035             : stdair::RootException (iWhat) {}
00036     };
00037
00041     class UnconstrainingException : public
stdair::RootException {
00042     public:
00044         UnconstrainingException (const std::string& iWhat)
00045             : stdair::RootException (iWhat) {}
00046     };
00047
00051     class ForecastException : public stdair::RootException {
00052     public:
00054         ForecastException (const std::string& iWhat)
00055             : stdair::RootException (iWhat) {}
00056     };
00057
00061     class OptimisationException : public
stdair::RootException {
00062     public:
00064         OptimisationException (const std::string& iWhat)
00065             : stdair::RootException (iWhat) {}
00066     };
00067

```

```

00068
00069 // ////////// Type definitions //////////
00073 typedef boost::shared_ptr<RMOL_Service> RMOL_ServicePtr_T;
00074
00076 typedef std::vector<stdair::NbOfRequests_T> UnconstrainedDemandVector_T
;
00077
00079 typedef std::vector<stdair::NbOfBookings_T> BookingVector_T;
00080
00082 typedef std::vector<stdair::Flag_T> FlagVector_T;
00083
00086 typedef std::map<stdair::BookingClass*, UnconstrainedDemandVector_T>
BookingClassUnconstrainedDemandVectorMap_T
;
00087
00090 typedef std::map<stdair::BookingClass*, stdair::NbOfRequests_T>
BookingClassUnconstrainedDemandMap_T;
00091
00093 typedef std::map<const stdair::DTD_T, double> FRAT5Curve_T;
00094
00095 }
00096 #endif // __RMOL_RMOL_TYPES_HPP

```

26.95 rmol/service/RMOL_Service.cpp File Reference

```

#include <cassert>
#include <boost/make_shared.hpp>
#include <stdair/stdair_inventory_types.hpp>
#include <stdair/basic/BasChronometer.hpp>
#include <stdair/basic/ContinuousAttributeLite.hpp>
#include <stdair/bom/BomManager.hpp>
#include <stdair/bom/BomRetriever.hpp>
#include <stdair/bom/BomRoot.hpp>
#include <stdair/bom/Inventory.hpp>
#include <stdair/bom/FlightDate.hpp>
#include <stdair/bom/LegCabin.hpp>
#include <stdair/bom/LegDate.hpp>
#include <stdair/bom/YieldFeatures.hpp>
#include <stdair/bom/AirportPair.hpp>
#include <stdair/bom/PosChannel.hpp>
#include <stdair/bom/DatePeriod.hpp>
#include <stdair/bom/TimePeriod.hpp>
#include <stdair/bom/AirlineClassList.hpp>
#include <stdair/basic/BasConst_Request.hpp>
#include <stdair/basic/BasConst_Inventory.hpp>
#include <stdair/bom/SegmentDate.hpp>
#include <stdair/bom/SegmentCabin.hpp>
#include <stdair/bom/BookingClass.hpp>
#include <stdair/bom/OnDDate.hpp>
#include <stdair/bom/OnDDateTypes.hpp>
#include <stdair/command/CmdBomManager.hpp>
#include <stdair/service/Logger.hpp>
#include <stdair/STDAIR_Service.hpp>
#include <rmol/basic/BasConst_RMOL_Service.hpp>
#include <rmol/factory/FacRmolServiceContext.hpp>
#include <rmol/command/InventoryParser.hpp>
#include <rmol/command/Optimiser.hpp>
#include <rmol/command/Forecaster.hpp>
#include <rmol/service/RMOL_ServiceContext.hpp>
#include <rmol/RMOL_Service.hpp>

```


Namespaces

- namespace [RMOL](#)

26.96 RMOL_Service.cpp

```

00001 // //////////////////////////////////////
00002 // Import section
00003 // //////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 // Boost
00007 #include <boost/make_shared.hpp>
00008 // StdAir
00009 #include <stdair/stdair_inventory_types.hpp>
00010 #include <stdair/basic/BasChronometer.hpp>
00011 #include <stdair/basic/ContinuousAttributeLite.hpp>
00012 #include <stdair/bom/BomManager.hpp>
00013 #include <stdair/bom/BomRetriever.hpp>
00014 #include <stdair/bom/BomRoot.hpp>
00015 #include <stdair/bom/Inventory.hpp>
00016 #include <stdair/bom/FlightDate.hpp>
00017 #include <stdair/bom/LegCabin.hpp>
00018 #include <stdair/bom/LegDate.hpp>
00019 #include <stdair/bom/YieldFeatures.hpp>
00020 #include <stdair/bom/AirportPair.hpp>
00021 #include <stdair/bom/PosChannel.hpp>
00022 #include <stdair/bom/DatePeriod.hpp>
00023 #include <stdair/bom/TimePeriod.hpp>
00024 #include <stdair/bom/AirlineClassList.hpp>
00025 #include <stdair/basic/BasConst_Request.hpp>
00026 #include <stdair/basic/BasConst_Inventory.hpp>
00027 #include <stdair/bom/Inventory.hpp>
00028 #include <stdair/bom/FlightDate.hpp>
00029 #include <stdair/bom/SegmentDate.hpp>
00030 #include <stdair/bom/SegmentCabin.hpp>
00031 #include <stdair/bom/BookingClass.hpp>
00032 #include <stdair/bom/OnDDate.hpp>
00033 #include <stdair/bom/OnDDateTypes.hpp>
00034 #include <stdair/command/CmdBomManager.hpp>
00035 #include <stdair/service/Logger.hpp>
00036 #include <stdair/STDAIR_Service.hpp>
00037 // RMOL
00038 #include <rmol/basic/BasConst_RMOL_Service.hpp>
00039 >
00039 #include <rmol/factory/FacRmolServiceContext.hpp>
00040 >
00040 #include <rmol/command/InventoryParser.hpp>
00041 #include <rmol/command/Optimiser.hpp>
00042 #include <rmol/command/Forecaster.hpp>
00043 #include <rmol/service/RMOL_ServiceContext.hpp>
00044 >
00044 #include <rmol/RMOL_Service.hpp>
00045
00046 namespace RMOL {
00047
00048 // //////////////////////////////////////
00049 RMOL_Service::RMOL_Service ()
00050 : _rmolServiceContext (NULL),
00051   _previousForecastDate (stdair::Date_T (2000, 1, 1)) {
00052     assert (false);
00053 }
00054
00055 // //////////////////////////////////////
00056 RMOL_Service::RMOL_Service (const RMOL_Service& iService) :
00057   _rmolServiceContext (NULL),
00058   _previousForecastDate (stdair::Date_T (2000, 1, 1)) {
00059     assert (false);
00060 }
00061
00062 // //////////////////////////////////////
00063 RMOL_Service::RMOL_Service (const stdair::BasLogParams& iLogParams) :
00064   _rmolServiceContext (NULL),
00065   _previousForecastDate (stdair::Date_T (2000, 1, 1)) {
00066
00067     // Initialise the STDAIR service handler
00068     stdair::STDAIR_ServicePtr_T lSTDAIR_Service_ptr =
00069       initStdAirService (iLogParams);
00070
00071     // Initialise the service context
00072     initServiceContext ();
00073
00074     // Add the StdAir service context to the RMOL service context

```

```

00075 // \note RMOL owns the STDAIR service resources here.
00076 const bool ownStdairService = true;
00077 addStdAirService (lSTDAIR_Service_ptr, ownStdairService);
00078
00079 // Initialise the (remaining of the) context
00080 initRmolService();
00081 }
00082
00083 // //////////////////////////////////////
00084 RMOL_Service::RMOL_Service (const stdair::BasLogParams& iLogParams,
00085                             const stdair::BasDBParams& iDBParams) :
00086     _rmolServiceContext (NULL),
00087     _previousForecastDate (stdair::Date_T (2000, 1, 1)) {
00088
00089     // Initialise the STDAIR service handler
00090     stdair::STDAIR_ServicePtr_T lSTDAIR_Service_ptr =
00091         initStdAirService (iLogParams, iDBParams);
00092
00093     // Initialise the service context
00094     initServiceContext();
00095
00096     // Add the StdAir service context to the RMOL service context
00097     // \note RMOL owns the STDAIR service resources here.
00098     const bool ownStdairService = true;
00099     addStdAirService (lSTDAIR_Service_ptr, ownStdairService);
00100
00101     // Initialise the (remaining of the) context
00102     initRmolService();
00103 }
00104
00105 // //////////////////////////////////////
00106 RMOL_Service::RMOL_Service (stdair::STDAIR_ServicePtr_T ioSTDAIRServicePtr)
00107 : _rmolServiceContext (NULL),
00108   _previousForecastDate (stdair::Date_T (2000, 1, 1)) {
00109
00110     // Initialise the context
00111     initServiceContext();
00112
00113     // Add the StdAir service context to the RMOL service context.
00114     // \note RMOL does not own the STDAIR service resources here.
00115     const bool doesNotOwnStdairService = false;
00116     addStdAirService (ioSTDAIRServicePtr, doesNotOwnStdairService);
00117
00118     // Initialise the (remaining of the) context
00119     initRmolService();
00120 }
00121
00122 // //////////////////////////////////////
00123 RMOL_Service::~RMOL_Service() {
00124     // Delete/Clean all the objects from memory
00125     finalise();
00126 }
00127
00128 // //////////////////////////////////////
00129 void RMOL_Service::finalise() {
00130     assert (_rmolServiceContext != NULL);
00131     // Reset the (Boost.)Smart pointer pointing on the STDAIR_Service object.
00132     _rmolServiceContext->reset();
00133 }
00134
00135 // //////////////////////////////////////
00136 void RMOL_Service::initServiceContext() {
00137     // Initialise the service context
00138     RMOL_ServiceContext& lRMOL_ServiceContext =
00139         FacRmolServiceContext::instance().create
00140 ();
00141     _rmolServiceContext = &lRMOL_ServiceContext;
00142 }
00143
00144 // //////////////////////////////////////
00145 void RMOL_Service::
00146 addStdAirService (stdair::STDAIR_ServicePtr_T ioSTDAIR_Service_ptr,
00147                  const bool iOwnStdairService) {
00148
00149     // Retrieve the RMOL service context
00150     assert (_rmolServiceContext != NULL);
00151     RMOL_ServiceContext& lRMOL_ServiceContext = *_rmolServiceContext;
00152
00153     // Store the STDAIR service object within the (AIRINV) service context
00154     lRMOL_ServiceContext.setSTDAIR_Service (ioSTDAIR_Service_ptr,
00155                                             iOwnStdairService);
00156 }
00157
00158 // //////////////////////////////////////
00159 stdair::STDAIR_ServicePtr_T RMOL_Service::
00160 initStdAirService (const stdair::BasLogParams& iLogParams) {

```

```

00168     stdair::STDAIR_ServicePtr_T lSTDAIR_Service_ptr =
00169         boost::make_shared<stdair::STDAIR_Service> (iLogParams);
00170
00171     return lSTDAIR_Service_ptr;
00172 }
00173
00174 // //////////////////////////////////////
00175 stdair::STDAIR_ServicePtr_T RMOL_Service::
00176     initStdAirService (const stdair::BasLogParams& iLogParams,
00177         const stdair::BasDBParams& iDBParams) {
00178
00186     stdair::STDAIR_ServicePtr_T lSTDAIR_Service_ptr =
00187         boost::make_shared<stdair::STDAIR_Service> (iLogParams, iDBParams);
00188
00189     return lSTDAIR_Service_ptr;
00190 }
00191
00192 // //////////////////////////////////////
00193 void RMOL_Service::initRmolService() {
00194     // Do nothing at this stage. A sample BOM tree may be built by
00195     // calling the buildSampleBom() method
00196 }
00197
00198 // //////////////////////////////////////
00199 void RMOL_Service::
00200     parseAndLoad (const stdair::CabinCapacity_T& iCabinCapacity,
00201         const stdair::Filename_T& iInputFileName) {
00202
00203     // Retrieve the RMOL service context
00204     if (_rmolServiceContext == NULL) {
00205         throw stdair::NonInitialisedServiceException ("The RMOL service has not"
00206             " been initialised");
00207     }
00208     assert (_rmolServiceContext != NULL);
00209     RMOL_ServiceContext& lRMOL_ServiceContext = *
00210         _rmolServiceContext;
00211
00212     // Retrieve the StdAir service object from the (RMOL) service context
00213     stdair::STDAIR_Service& lSTDAIR_Service =
00214         lRMOL_ServiceContext.getSTDAIR_Service();
00215     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
00216
00217     // Build a dummy inventory with a leg-cabin which has the given capacity.
00218     lSTDAIR_Service.buildDummyInventory (iCabinCapacity);
00219
00220     // Complete the BOM tree with the optimisation problem specification
00221     InventoryParser::parseInputFileAndBuildBom
00222         (iInputFileName, lBomRoot);
00223 }
00224
00225 // //////////////////////////////////////
00226 void RMOL_Service::buildSampleBom() {
00227
00228     // Retrieve the RMOL service context
00229     if (_rmolServiceContext == NULL) {
00230         throw stdair::NonInitialisedServiceException ("The RMOL service has not"
00231             " been initialised");
00232     }
00233     assert (_rmolServiceContext != NULL);
00234
00235     // Retrieve the RMOL service context and whether it owns the Stdair
00236     // service
00237     RMOL_ServiceContext& lRMOL_ServiceContext = *
00238         _rmolServiceContext;
00239     const bool doesOwnStdairService =
00240         lRMOL_ServiceContext.getOwnStdairServiceFlag();
00241
00242     // Retrieve the StdAir service object from the (RMOL) service context
00243     stdair::STDAIR_Service& lSTDAIR_Service =
00244         lRMOL_ServiceContext.getSTDAIR_Service();
00245
00246     if (doesOwnStdairService == true) {
00247         //
00248         lSTDAIR_Service.buildSampleBom();
00249     }
00250 }
00251
00252 // //////////////////////////////////////
00253 void RMOL_Service::optimalOptimisationByMCIntegration
00254     (const int K) {
00255     assert (_rmolServiceContext != NULL);
00256     RMOL_ServiceContext& lRMOL_ServiceContext = *
00257         _rmolServiceContext;
00258
00259     // Retrieve the StdAir service
00260     stdair::STDAIR_Service& lSTDAIR_Service =

```

```

00278     lRMOL_ServiceContext.getSTDAIR_Service();
00279     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
00280
00281     //
00282     stdair::LegCabin& lLegCabin =
00283         stdair::BomRetriever::retrieveDummyLegCabin (lBomRoot);
00284
00285     stdair::BasChronometer lOptimisationChronometer;
00286     lOptimisationChronometer.start();
00287
00288     Optimiser::optimalOptimisationByMCIntegration
00289     (K, lLegCabin);
00290
00291     const double lOptimisationMeasure = lOptimisationChronometer.elapsed();
00292
00293     // DEBUG
00294     STDAIR_LOG_DEBUG ("Optimisation by Monte-Carlo performed in "
00295         << lOptimisationMeasure);
00296     STDAIR_LOG_DEBUG ("Result: " << lLegCabin.displayVirtualClassList());
00297
00298     std::ostringstream logStream;
00299     stdair::BidPriceVector_T lBidPriceVector = lLegCabin.getBidPriceVector();
00300     logStream << "Bid-Price Vector (BPV): ";
00301     unsigned int size = lBidPriceVector.size();
00302
00303     for (unsigned int i = 0; i < size - 1; ++i) {
00304         const double bidPrice = lBidPriceVector.at(i);
00305         logStream << std::fixed << std::setprecision (2) << bidPrice << ", ";
00306     }
00307     const double bidPrice = lBidPriceVector.at(size - 1);
00308     logStream << std::fixed << std::setprecision (2) << bidPrice;
00309     STDAIR_LOG_DEBUG (logStream.str());
00310 }
00311 // //////////////////////////////////////
00312 void RMOL_Service::optimalOptimisationByDP
00313 () {
00314 }
00315 // //////////////////////////////////////
00316 void RMOL_Service::heuristicOptimisationByEmsr
00317 () {
00318     assert (_rmolServiceContext != NULL);
00319     RMOL_ServiceContext& lRMOL_ServiceContext = *
00320     _rmolServiceContext;
00321
00322     // Retrieve the StdAir service
00323     stdair::STDAIR_Service& lSTDAIR_Service =
00324         lRMOL_ServiceContext.getSTDAIR_Service();
00325     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
00326
00327     //
00328     stdair::LegCabin& lLegCabin =
00329         stdair::BomRetriever::retrieveDummyLegCabin (lBomRoot);
00330
00331     stdair::BasChronometer lOptimisationChronometer;
00332     lOptimisationChronometer.start();
00333
00334     Optimiser::heuristicOptimisationByEmsr
00335     (lLegCabin);
00336
00337     const double lOptimisationMeasure = lOptimisationChronometer.elapsed();
00338
00339     // DEBUG
00340     STDAIR_LOG_DEBUG ("Optimisation EMSR performed in "
00341         << lOptimisationMeasure);
00342     STDAIR_LOG_DEBUG ("Result: " << lLegCabin.displayVirtualClassList());
00343
00344     stdair::BidPriceVector_T lBidPriceVector = lLegCabin.getBidPriceVector();
00345     std::ostringstream logStream;
00346     logStream << "Bid-Price Vector (BPV): ";
00347     unsigned int idx = 0;
00348     for (stdair::BidPriceVector_T::const_iterator itBP = lBidPriceVector.begin()
00349 );
00350         itBP != lBidPriceVector.end(); ++itBP) {
00351         if (idx != 0) {
00352             logStream << ", ";
00353         }
00354         const stdair::BidPrice_T& lBidPrice = *itBP;
00355         logStream << std::fixed << std::setprecision (2) << lBidPrice;
00356     }
00357     // DEBUG
00358     STDAIR_LOG_DEBUG (logStream.str());
00359 }
00360 // //////////////////////////////////////
00361 void RMOL_Service::heuristicOptimisationByEmsrA
00362 () {

```

```

00358     assert (_rmolServiceContext != NULL);
00359     RMOL_ServiceContext& lRMOL_ServiceContext = *
_rmolServiceContext;
00360
00361     // Retrieve the StdAir service
00362     stdair::STDAIR_Service& lSTDAIR_Service =
00363         lRMOL_ServiceContext.getSTDAIR_Service();
00364     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
00365
00366     //
00367     stdair::LegCabin& lLegCabin =
00368         stdair::BomRetriever::retrieveDummyLegCabin (lBomRoot);
00369
00370     Optimiser::heuristicOptimisationByEmsrA
(lLegCabin);
00371
00372     // DEBUG
00373     STDAIR_LOG_DEBUG ("Result: " << lLegCabin.displayVirtualClassList());
00374
00375 }
00376
00377 // //////////////////////////////////////
00378 void RMOL_Service::heuristicOptimisationByEmsrB
() {
00379     assert (_rmolServiceContext != NULL);
00380     RMOL_ServiceContext& lRMOL_ServiceContext = *
_rmolServiceContext;
00381
00382     // Retrieve the StdAir service
00383     stdair::STDAIR_Service& lSTDAIR_Service =
00384         lRMOL_ServiceContext.getSTDAIR_Service();
00385     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
00386
00387     //
00388     stdair::LegCabin& lLegCabin =
00389         stdair::BomRetriever::retrieveDummyLegCabin (lBomRoot);
00390
00391     Optimiser::heuristicOptimisationByEmsrB
(lLegCabin);
00392
00393     // DEBUG
00394     STDAIR_LOG_DEBUG ("Result: " << lLegCabin.displayVirtualClassList());
00395 }
00396
00397 // //////////////////////////////////////
00398 bool RMOL_Service::
00399 optimise (stdair::FlightDate& ioFlightDate,
00400          const stdair::DateTime_T& iRMEEventTime,
00401          const stdair::ForecastingMethod& iForecastingMethod,
00402          const stdair::PartnershipTechnique& iPartnershipTechnique) {
00403
00404
00405     STDAIR_LOG_DEBUG ("Forecast & Optimisation");
00406
00407     const stdair::PartnershipTechnique::EN_PartnershipTechnique&
lPartnershipTechnique =
00408         iPartnershipTechnique.getTechnique();
00409
00410     switch (lPartnershipTechnique) {
00411     case stdair::PartnershipTechnique::RAE_DA:
00412     case stdair::PartnershipTechnique::IBP_DA:{
00413         if (_previousForecastDate < iRMEEventTime.date()) {
00414             forecastOnD (iRMEEventTime);
00415             resetDemandInformation (iRMEEventTime);
00416             projectAggregatedDemandOnLegCabins (
iRMEEventTime);
00417             optimiseOnD (iRMEEventTime);
00418         }
00419         break;
00420     }
00421     case stdair::PartnershipTechnique::RAE_YP:
00422     case stdair::PartnershipTechnique::IBP_YP:
00423     case stdair::PartnershipTechnique::IBP_YP_U:{
00424         if (_previousForecastDate < iRMEEventTime.date()) {
00425             forecastOnD (iRMEEventTime);
00426             resetDemandInformation (iRMEEventTime);
00427             projectOnDDemandOnLegCabinsUsingYP (
iRMEEventTime);
00428             optimiseOnD (iRMEEventTime);
00429         }
00430         break;
00431     }
00432     case stdair::PartnershipTechnique::RMC:{
00433         if (_previousForecastDate < iRMEEventTime.date()) {
00434             forecastOnD (iRMEEventTime);
00435             resetDemandInformation (iRMEEventTime);
00436             updateBidPrice (iRMEEventTime);

```

```

00437         projectOnDDemandOnLegCabinsUsingDYP
00438         (iRMEventTime);
00439         optimiseOnDUsingRMCooperation (
00440         iRMEventTime);
00441     }
00442     break;
00443 }
00444 case stdair::PartnershipTechnique::A_RMC:{
00445     if (_previousForecastDate < iRMEventTime.date()) {
00446         forecastOnD (iRMEventTime);
00447         resetDemandInformation (iRMEventTime);
00448         updateBidPrice (iRMEventTime);
00449         projectOnDDemandOnLegCabinsUsingDYP
00450         (iRMEventTime);
00451         optimiseOnDUsingAdvancedRMCooperation
00452         (iRMEventTime);
00453     }
00454     break;
00455 }
00456 case stdair::PartnershipTechnique::NONE:{
00457     // DEBUG
00458     STDAIR_LOG_DEBUG ("Forecast");
00459     // 1. Forecast
00460     bool isForecasted = false;
00461     const stdair::ForecastingMethod& lForecastingMethod
00462     =
00463     iForecastingMethod.getMethod();
00464     switch (lForecastingMethod) {
00465     case stdair::ForecastingMethod::ADD_PK: {
00466         isForecasted = Forecaster::forecastUsingAdditivePickUp
00467         (ioFlightDate,
00468         iRMEventTime);
00469         break;
00470     }
00471     case stdair::ForecastingMethod::MUL_PK: {
00472         isForecasted =
00473         Forecaster::forecastUsingMultiplicativePickUp
00474         (ioFlightDate,
00475         iRMEventTime);
00476         break;
00477     }
00478     default: {
00479         assert (false);
00480         break;
00481     }
00482 }
00483 // DEBUG
00484 STDAIR_LOG_DEBUG ("Forecast successful: " << isForecasted);
00485 // 2. Optimisation
00486 if (isForecasted == true) {
00487     // DEBUG
00488     STDAIR_LOG_DEBUG ("Optimise");
00489     Optimiser::optimise (ioFlightDate);
00490     return true;
00491 }
00492 break;
00493 default:{
00494     assert (false);
00495     break;
00496 }
00497 }
00498 return false;
00499 }
00500 ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00501 void RMOL_Service::forecastOnD (const
00502 stdair::DateTime_T& iRMEventTime) {
00503     if (_rmolServiceContext == NULL) {
00504         throw stdair::NonInitialisedServiceException ("The Rmol service "
00505         "has not been initialised");
00506     }
00507     assert (_rmolServiceContext != NULL);
00508     RMOL_ServiceContext& lRMOL_ServiceContext = *
00509     _rmolServiceContext;
00510     // Retrieve the bom root
00511     stdair::STDAIR_Service& lSTDAIR_Service =
00512     lRMOL_ServiceContext.getSTDAIR_Service();
00513     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();

```

```

00514 // Retrieve the date from the RM event
00515 const stdair::Date_T lDate = iRMEventTime.date();
00516
00517 _previousForecastDate = lDate;
00518
00519 const stdair::InventoryList_T& lInventoryList =
00520     stdair::BomManager::getList<stdair::Inventory> (lBomRoot);
00521 assert (!lInventoryList.empty());
00522 for (stdair::InventoryList_T::const_iterator itInv = lInventoryList.begin()
;
00523     itInv != lInventoryList.end(); ++itInv) {
00524     const stdair::Inventory* lInventory_ptr = *itInv;
00525     assert (lInventory_ptr != NULL);
00526     if (stdair::BomManager::hasList<stdair::OnDDate> (*lInventory_ptr)) {
00527         const stdair::OnDDateList_T lOnDDateList =
00528             stdair::BomManager::getList<stdair::OnDDate> (*lInventory_ptr);
00529
00530         for (stdair::OnDDateList_T::const_iterator itOD = lOnDDateList.begin();
00531             itOD != lOnDDateList.end(); ++itOD) {
00532             stdair::OnDDate* lOnDDate_ptr = *itOD;
00533             assert (lOnDDate_ptr != NULL);
00534
00535             const stdair::Date_T lDepartureDate = lOnDDate_ptr->getDate();
00536             stdair::DateOffset_T lDateOffset = lDepartureDate - lDate;
00537             stdair::DTD_T lDTD = short (lDateOffset.days());
00538
00539             stdair::DCPLIST_T::const_iterator itDCP =
00540                 std::find (stdair::DEFAULT_DCP_LIST.begin()
00541 ,
00542                             stdair::DEFAULT_DCP_LIST.end(),
00543                             lDTD);
00544             // Check if the forecast for this O&D date needs to be forecasted.
00545             if (itDCP != stdair::DEFAULT_DCP_LIST.end())
00546             {
00547                 // Retrieve the total forecast map.
00548                 const stdair::CabinForecastMap_T& lTotalForecastMap =
00549                     lOnDDate_ptr->getTotalForecastMap();
00550
00551                 // Browse the map and make a forecast for every cabin.
00552                 for (stdair::CabinForecastMap_T::const_iterator itCF =
00553                     lTotalForecastMap.begin();
00554                     itCF != lTotalForecastMap.end(); ++itCF) {
00555                     const stdair::CabinCode_T lCabinCode = itCF->first;
00556                     stdair::YieldFeatures* lYieldFeatures_ptr =
00557                         getYieldFeatures(*lOnDDate_ptr, lCabinCode,
00558 lBomRoot);
00559                     if (lYieldFeatures_ptr == NULL) {
00560                         STDAIR_LOG_ERROR ("Cannot find yield corresponding to "
00561 << "the O&D date"
00562 << lOnDDate_ptr->toString()
00563 << " Cabin " << lCabinCode);
00564                     }
00565                     assert (false);
00566                     forecastOnD (*lYieldFeatures_ptr, *lOnDDate_ptr,
00567 lCabinCode, lDTD,
00568                             lBomRoot);
00569                 }
00570             }
00571         }
00572     }
00573 }
00574
00575 // //////////////////////////////////////
00576 stdair::YieldFeatures* RMOL_Service::
00577 getYieldFeatures(const stdair::OnDDate& iOnDDate,
00578                 const stdair::CabinCode_T& iCabinCode,
00579                 stdair::BomRoot& iBomRoot) {
00580
00581     const stdair::AirportCode_T& lOrigin = iOnDDate.getOrigin();
00582     const stdair::AirportCode_T& lDestination = iOnDDate.getDestination();
00583
00584     const stdair::Date_T& lDepartureDate = iOnDDate.getDate();
00585
00586     // Build the airport pair key out of O&D and get the airport pair object
00587     const stdair::AirportPairKey lAirportPairKey(lOrigin, lDestination);
00588     stdair::AirportPair* lAirportPair_ptr = stdair::BomManager::
00589         getObjectPtr<stdair::AirportPair> (iBomRoot,
00590 lAirportPairKey.toString());
00591     if (lAirportPair_ptr == NULL) {
00592         STDAIR_LOG_ERROR ("Cannot find yield corresponding to the airport "
00593 << "pair: " << lAirportPairKey.toString());
00594     }
00595     assert (false);
00596 }
00597
00598 // Retrieve the corresponding date period to lDepartureDate.
00599 const stdair::DatePeriodList_T lDatePeriodList =

```

```

00595     stdair::BomManager::getList<stdair::DatePeriod> (*lAirportPair_ptr);
00596 for (stdair::DatePeriodList_T::const_iterator itDatePeriod =
00597     lDatePeriodList.begin();
00598     itDatePeriod != lDatePeriodList.end(); ++itDatePeriod) {
00599     const stdair::DatePeriod* lDatePeriod_ptr = *itDatePeriod;
00600     assert (lDatePeriod_ptr != NULL);
00601
00602     const bool isDepartureDateValid =
00603         lDatePeriod_ptr->isDepartureDateValid (lDepartureDate);
00604
00605     if (isDepartureDateValid == true) {
00606         // Retrieve the PoS-Channel.
00607         // TODO: Use POS and Channel from demand instead of default
00608         const stdair::PosChannelKey lPosChannelKey (stdair::DEFAULT_POS,
00609             stdair::DEFAULT_CHANNEL);
00610         stdair::PosChannel* lPosChannel_ptr = stdair::BomManager::
00611             getObjectPtr<stdair::PosChannel> (*lDatePeriod_ptr,
00612                 lPosChannelKey.toString());
00613         if (lPosChannel_ptr == NULL) {
00614             STDAIR_LOG_ERROR ("Cannot find yield corresponding to the PoS-"
00615                 "<< \"Channel: \" << lPosChannelKey.toString());
00616             assert (false);
00617         }
00618         // Retrieve the yield features.
00619         const stdair::TimePeriodList_T lTimePeriodList = stdair::
00620             BomManager::getList<stdair::TimePeriod> (*lPosChannel_ptr);
00621         for (stdair::TimePeriodList_T::const_iterator itTimePeriod =
00622             lTimePeriodList.begin();
00623             itTimePeriod != lTimePeriodList.end(); ++itTimePeriod) {
00624             const stdair::TimePeriod* lTimePeriod_ptr = *itTimePeriod;
00625             assert (lTimePeriod_ptr != NULL);
00626
00627             // TODO: Use trip type from demand instead of default value.
00628             const stdair::YieldFeaturesKey lYieldFeaturesKey (
00629                 stdair::TRIP_TYPE_ONE_WAY,
00630                     iCabinCode);
00631             stdair::YieldFeatures* oYieldFeatures_ptr = stdair::BomManager::
00632                 getObjectPtr<stdair::YieldFeatures> (*lTimePeriod_ptr,
00633                     lYieldFeaturesKey.toString());
00634             if (oYieldFeatures_ptr != NULL) {
00635                 return oYieldFeatures_ptr;
00636             }
00637         }
00638     }
00639     return NULL;
00640 }
00641
00642
00643
00644 // //////////////////////////////////////
00645 void RMOL_Service::
00646 forecastOnD (const stdair::YieldFeatures& iYieldFeatures,
00647     stdair::OnDDate& iOnDDate,
00648     const stdair::CabinCode_T& iCabinCode,
00649     const stdair::DTD_T& iDTD,
00650     stdair::BomRoot& iBomRoot) {
00651
00652     const stdair::AirlineClassListList_T lAirlineClassListList =
00653         stdair::BomManager::getList<stdair::AirlineClassList> (iYieldFeatures);
00654     assert (lAirlineClassListList.begin() != lAirlineClassListList.end());
00655
00656     // Yield order check
00657     stdair::AirlineClassListList_T::const_iterator itACL =
00658         lAirlineClassListList.begin();
00659     stdair::Yield_T lPreviousYield ((*itACL)->getYield());
00660     ++itACL;
00661     for (; itACL != lAirlineClassListList.end(); ++itACL) {
00662         const stdair::AirlineClassList* lAirlineClassList = *itACL;
00663         const stdair::Yield_T& lYield = lAirlineClassList->getYield();
00664         if (lYield <= lPreviousYield) {
00665             lPreviousYield = lYield;
00666         }
00667         else{
00668             STDAIR_LOG_ERROR ("Yields should be given in a descendant order"
00669                 "<< \" in the yield input file\" );
00670             assert (false);
00671         }
00672     }
00673     // Proportion factor list initialisation
00674     // Each element corresponds to a yield rule
00675     stdair::ProportionFactorList_T lProportionFactorList;
00676     stdair::ProportionFactor_T lPreviousProportionFactor = 0;
00677
00678     // Retrieve the minimal willingness to pay associated to the demand
00679     const stdair::WTPDemandPair_T& lTotalForecast =
00680         iOnDDate.getTotalForecast (iCabinCode);

```



```

00681     const stdair::WTP_T& lMinWTP = lTotalForecast.first;
00682
00683     // Retrieve the remaining percentage of booking requests
00684     const stdair::ContinuousAttributeLite<stdair::FloatDuration_T>
00685         lArrivalPattern (stdair::DEFAULT_DTD_PROB_MAP);
00686
00687     STDAIR_LOG_DEBUG (lArrivalPattern.displayCumulativeDistribution());
00688     const stdair::Probability_T lRemainingProportion =
00689         lArrivalPattern.getRemainingProportion(-float(iDTD));
00690
00691     // Compute the characteristics (mean and std dev) of the total
00692     // forecast demand to come
00693     const stdair::MeanStdDevPair_T lForecatsMeanStdDevPair =
00694         lTotalForecast.second;
00695     const stdair::MeanValue_T& lMeanValue =
00696         lForecatsMeanStdDevPair.first;
00697     const stdair::MeanValue_T& lRemainingMeanValue =
00698         lRemainingProportion*lMeanValue;
00699     const stdair::StdDevValue_T& lStdDevValue =
00700         lForecatsMeanStdDevPair.second;
00701     const stdair::StdDevValue_T& lRemainingStdDevValue =
00702         lRemainingProportion*lStdDevValue;
00703
00704     // Retrieve the frat5 coef corresponding to the input dtd
00705     stdair::DTDFratMap_T::const_iterator itDFC =
00706         stdair::DEFAULT_DTD_FRAT5COEF_MAP.find(iDTD);
00707     if (itDFC == stdair::DEFAULT_DTD_FRAT5COEF_MAP.end()) {
00708         STDAIR_LOG_ERROR ("Cannot find frat5 coef for DTD = " << iDTD );
00709         assert (false);
00710     }
00711     stdair::RealNumber_T lFrat5Coef =
00712         stdair::DEFAULT_DTD_FRAT5COEF_MAP.at(iDTD);
00713
00714     STDAIR_LOG_DEBUG ("Remaining proportion " << lRemainingProportion
00715         << " Total " << lMeanValue
00716         << " StdDev " << lStdDevValue
00717         << "Frat5 Coef " << lFrat5Coef);
00718
00719     std::ostringstream oStr;
00720     // Compute the "forecast demand to come" proportion by class
00721     itACL = lAirlineClassListList.begin();
00722     for (; itACL != lAirlineClassListList.end(); ++itACL) {
00723         const stdair::AirlineClassList* lAirlineClassList_ptr = *itACL;
00724         const stdair::Yield_T& lYield = lAirlineClassList_ptr->getYield();
00725         stdair::ProportionFactor_T lProportionFactor =
00726             exp ((lYield - lMinWTP)*log(0.5)/(lMinWTP*(lFrat5Coef-1.0)));
00727         // If the yield is smaller than minimal WTP, the factor is greater than
00728         1.
00729         // In that case it should be modified and put to 1.
00730         lProportionFactor = std::min (lProportionFactor, 1.0);
00731         lProportionFactorList.push_back(lProportionFactor -
00732             lPreviousProportionFactor);
00733         lPreviousProportionFactor = lProportionFactor;
00734         oStr << lAirlineClassList_ptr->toString() << lProportionFactor << " ";
00735     }
00736     STDAIR_LOG_DEBUG (oStr.str());
00737
00738     // Sanity check
00739     assert (lAirlineClassListList.size() == lProportionFactorList.size());
00740
00741     STDAIR_LOG_DEBUG ("Forecast for " << iOnDDate.describeKey()
00742         << " " << iDTD << " days to departure");
00743
00744     // store the forecast demand to come characteristics in the booking classes
00745     stdair::ProportionFactorList_T::const_iterator itPF =
00746         lProportionFactorList.begin();
00747     itACL = lAirlineClassListList.begin();
00748     for (; itACL != lAirlineClassListList.end(); ++itACL, ++itPF) {
00749         const stdair::AirlineClassList* lAirlineClassList_ptr = *itACL;
00750         const stdair::ProportionFactor_T& lProportionFactor = *itPF;
00751         stdair::MeanValue_T lMeanValue = lProportionFactor*lRemainingMeanValue;
00752         stdair::StdDevValue_T lStdDevValue =
00753             lProportionFactor*lRemainingStdDevValue;
00754         setOnDForecast(*lAirlineClassList_ptr, lMeanValue,
00755             lStdDevValue,
00756             iOnDDate, iCabinCode, iBomRoot);
00757     }
00758
00759     // //////////////////////////////////////
00760     void RMOL_Service::
00761     setOnDForecast (const stdair::AirlineClassList&
00762         iAirlineClassList,
00763         const stdair::MeanValue_T& iMeanValue,
00764         const stdair::StdDevValue_T& iStdDevValue,

```

```

00764         stdair::OnDDate& iOnDDate,
00765         const stdair::CabinCode_T& iCabinCode,
00766         stdair::BomRoot& iBomRoot) {
00767
00768     const stdair::AirportCode_T& lOrigin = iOnDDate.getOrigin();
00769     const stdair::AirportCode_T& lDestination = iOnDDate.getDestination();
00770
00771     const stdair::Date_T& lDepartureDate = iOnDDate.getDate();
00772
00773     const stdair::AirlineCodeList_T& lAirlineCodeList =
00774         iAirlineClassList.getAirlineCodeList();
00775
00776     // Retrieve the class list (one class per airline)
00777     const stdair::ClassList_StringList_T& lClassList_StringList =
00778         iAirlineClassList.getClassCodeList();
00779     assert (!lClassList_StringList.empty());
00780     stdair::ClassCodeList_T lClassCodeList;
00781     for (stdair::ClassList_StringList_T::const_iterator itCL =
00782         lClassList_StringList.begin();
00783         itCL != lClassList_StringList.end(); ++itCL){
00784         const stdair::ClassList_String_T& lClassList_String = *itCL;
00785         assert (lClassList_String.size() > 0);
00786         stdair::ClassCode_T lFirstClass;
00787         lFirstClass.append (lClassList_String, 0, 1);
00788         lClassCodeList.push_back(lFirstClass);
00789     }
00790
00791     // Sanity check
00792     assert (lAirlineCodeList.size() == lClassCodeList.size());
00793     assert (!lAirlineCodeList.empty());
00794
00795     if (lAirlineCodeList.size() == 1) {
00796         // Store the forecast information in the case of a single segment
00797         stdair::AirlineCode_T lAirlineCode = lAirlineCodeList.front();
00798         stdair::ClassCode_T lClassCode = lClassCodeList.front();
00799         stdair::Yield_T lYield = iAirlineClassList.getYield();
00800         setOnDForecast(lAirlineCode, lDepartureDate, lOrigin,
00801             lDestination, iCabinCode, lClassCode,
00802             iMeanValue, iStdDevValue, lYield, iBomRoot);
00803     } else {
00804         // Store the forecast information in the case of a multiple segment
00805
00806         stdair::Yield_T lYield = iAirlineClassList.getYield();
00807         for (stdair::AirlineCodeList_T::const_iterator itAC =
00808             lAirlineCodeList.begin();
00809             itAC != lAirlineCodeList.end(); ++itAC) {
00810             const stdair::AirlineCode_T& lAirlineCode = *itAC;
00811             setOnDForecast(lAirlineCodeList, lAirlineCode,
00812                 lDepartureDate, lOrigin,
00813                 lDestination, iCabinCode, lClassCodeList,
00814                 iMeanValue, iStdDevValue, lYield, iBomRoot);
00815         }
00816     }
00817
00818     // //////////////////////////////////////
00819     void RMOL_Service::
00820     setOnDForecast (const stdair::AirlineCode_T& iAirlineCode,
00821         const stdair::Date_T& iDepartureDate,
00822         const stdair::AirportCode_T& iOrigin,
00823         const stdair::AirportCode_T& iDestination,
00824         const stdair::CabinCode_T& iCabinCode,
00825         const stdair::ClassCode_T& iClassCode,
00826         const stdair::MeanValue_T& iMeanValue,
00827         const stdair::StdDevValue_T& iStdDevValue,
00828         const stdair::Yield_T& iYield,
00829         stdair::BomRoot& iBomRoot) {
00830     stdair::Inventory* lInventory_ptr = iBomRoot.getInventory(iAirlineCode);
00831     if (lInventory_ptr == NULL) {
00832         STDAIR_LOG_ERROR ("Cannot find the inventory corresponding"
00833             "<< " to the airline" << iAirlineCode) ;
00834         assert(false);
00835     }
00836     const stdair::OnDDateList_T lOnDDateList =
00837         stdair::BomManager::getList<stdair::OnDDate> (*lInventory_ptr);
00838     assert (!lOnDDateList.empty());
00839     bool lFoundOnDDate = false;
00840     for (stdair::OnDDateList_T::const_iterator itOD = lOnDDateList.begin();
00841         itOD != lOnDDateList.end(); ++itOD) {
00842         stdair::OnDDate* lOnDDate_ptr = *itOD;
00843         assert (lOnDDate_ptr != NULL);
00844         const stdair::Date_T& lDepartureDate = lOnDDate_ptr->getDate();
00845         const stdair::AirportCode_T& lOrigin = lOnDDate_ptr->getOrigin();
00846         const stdair::AirportCode_T& lDestination = lOnDDate_ptr->getDestination()
00847     );
00848     if (!stdair::BomManager::hasList<stdair::SegmentDate> (*lOnDDate_ptr)) {
00849         STDAIR_LOG_ERROR ("The O&D date " << lOnDDate_ptr->describeKey()

```

```

00849                                     << "has not been correctly initialized : SegmentDate
list is missing");
00850         assert (false);
00851     }
00852     const stdair::SegmentDateList_T& lSegmentDateList =
00853         stdair::BomManager::getList<stdair::SegmentDate> (*lOnDDate_ptr);
00854     // Check if the the O&D date is the one we are looking for
00855     if (lDepartureDate == iDepartureDate && lOrigin == iOrigin &&
00856         lDestination == iDestination && lSegmentDateList.size() == 1) {
00857         stdair::CabinClassPair_T lCabinClassPair (iCabinCode, iClassCode);
00858         stdair::CabinClassPairList_T lCabinClassPairList;
00859         lCabinClassPairList.push_back(lCabinClassPair);
00860         const stdair::MeanStdDevPair_T lMeanStdDevPair (iMeanValue,
iStdDevValue);
00861         const stdair::WTPDemandPair_T lWTPDemandPair (iYield, lMeanStdDevPair);
00862         lOnDDate_ptr->setDemandInformation(lCabinClassPairList, lWTPDemandPair)
;
00863         lFoundOnDDate = true;
00864         STDAIR_LOG_DEBUG (iAirlineCode << " Class " << iClassCode
00865             << " Mean " << iMeanValue
00866             << " Std Dev " << iStdDevValue);
00867         break;
00868     }
00869 }
00870
00871 if (!lFoundOnDDate) {
00872     STDAIR_LOG_ERROR ("Cannot find class " << iClassCode << " in cabin "
00873         << iCabinCode << " for the segment "
00874         << iOrigin << "-" << iDestination << " with"
00875         << " the airline " << iAirlineCode);
00876     assert(false);
00877 }
00878 }
00879
00880 // //////////////////////////////////////
00881 void RMOL_Service::
00882 setOnDDForecast (const stdair::AirlineCodeList_T&
iAirlineCodeList,
00883     const stdair::AirlineCode_T& iAirlineCode,
00884     const stdair::Date_T& iDepartureDate,
00885     const stdair::AirportCode_T& iOrigin,
00886     const stdair::AirportCode_T& iDestination,
00887     const stdair::CabinCode_T& iCabinCode,
00888     const stdair::ClassCodeList_T& iClassCodeList,
00889     const stdair::MeanValue_T& iMeanValue,
00890     const stdair::StdDevValue_T& iStdDevValue,
00891     const stdair::Yield_T& iYield,
00892     stdair::BomRoot& iBomRoot) {
00893     stdair::Inventory* lInventory_ptr = iBomRoot.getInventory(iAirlineCode);
00894     if (lInventory_ptr == NULL) {
00895         STDAIR_LOG_ERROR ("Cannot find the inventory corresponding"
00896             << " to the airline" << iAirlineCode) ;
00897         assert(false);
00898     }
00899     const stdair::OnDDateList_T lOnDDateList =
00900         stdair::BomManager::getList<stdair::OnDDate> (*lInventory_ptr);
00901     assert (!lOnDDateList.empty());
00902     bool lFoundOnDDate = false;
00903     for (stdair::OnDDateList_T::const_iterator itOD = lOnDDateList.begin();
00904         itOD != lOnDDateList.end(); ++itOD) {
00905         stdair::OnDDate* lOnDDate_ptr = *itOD;
00906         assert (lOnDDate_ptr != NULL);
00907         const stdair::Date_T& lDepartureDate = lOnDDate_ptr->getDate();
00908         const stdair::AirportCode_T& lOrigin = lOnDDate_ptr->getOrigin();
00909         const stdair::AirportCode_T& lDestination = lOnDDate_ptr->getDestination()
);
00910         if (!stdair::BomManager::hasList<stdair::SegmentDate> (*lOnDDate_ptr)) {
00911             STDAIR_LOG_ERROR ("The O&D date " << lOnDDate_ptr->describeKey()
00912                 << "has not been correctly initialized : SegmentDate
list is missing");
00913             assert (false);
00914         }
00915         const stdair::SegmentDateList_T& lSegmentDateList =
00916             stdair::BomManager::getList<stdair::SegmentDate> (*lOnDDate_ptr);
00917         // Check if the O&D date might be the one we are looking for.
00918         // There still is a test to go through to see if the combination of
airlines is right.
00919         if (lDepartureDate == iDepartureDate && lOrigin == iOrigin &&
00920             lDestination == iDestination && lSegmentDateList.size() ==
iAirlineCodeList.size()) {
00921             const stdair::SegmentDateList_T& lSegmentDateList =
00922                 stdair::BomManager::getList<stdair::SegmentDate> (*lOnDDate_ptr);
00923             stdair::AirlineCodeList_T::const_iterator itAC = iAirlineCodeList.begin
();
00924             stdair::SegmentDateList_T::const_iterator itSD = lSegmentDateList.begin

```

```

    );
00925     for (; itAC != iAirlineCodeList.end(); ++itAC, ++itSD) {
00926         const stdair::AirlineCode_T lForecastAirlineCode = *itAC;
00927         const stdair::SegmentDate* lSegmentDate_ptr = *itSD;
00928         // Check if the operating airline is a different one and check if it
00929         // is the airline that we are looking for.
00930         const stdair::SegmentDate* lOperatingSegmentDate_ptr =
00931             lSegmentDate_ptr->getOperatingSegmentDate ();
00932         if (lOperatingSegmentDate_ptr != NULL) {
00933             const stdair::FlightDate* lOperatingFD_ptr =
00934                 stdair::BomManager::getParentPtr<stdair::FlightDate>
00935                 (*lOperatingSegmentDate_ptr);
00936             const stdair::AirlineCode_T lOperatingAirlineCode =
00937                 lOperatingFD_ptr->getAirlineCode();
00938             if (lOperatingAirlineCode != lForecastAirlineCode) {
00939                 break;
00940             } else {
00941                 const stdair::AirlineCode_T lOperatingAirlineCode =
00942                     lOnDDate_ptr->getAirlineCode();
00943                 if (lOperatingAirlineCode != lForecastAirlineCode) {
00944                     break;
00945                 }
00946             }
00947         }
00948         if (itAC == iAirlineCodeList.end()) {lFoundOnDDate = true;}
00949     }
00950     if (lFoundOnDDate) {
00951         stdair::CabinClassPairList_T lCabinClassPairList;
00952         for (stdair::ClassCodeList_T::const_iterator itCC = iClassCodeList.
begin();
00953             itCC != iClassCodeList.end(); ++itCC) {
00954             const stdair::ClassCode_T lClassCode = *itCC;
00955             stdair::CabinClassPair_T lCabinClassPair (iCabinCode, lClassCode);
00956             lCabinClassPairList.push_back(lCabinClassPair);
00957         }
00958         const stdair::MeanStdDevPair_T lMeanStdDevPair (iMeanValue,
iStdDevValue);
00959         const stdair::YieldDemandPair_T lYieldDemandPair (iYield,
lMeanStdDevPair);
00960         lOnDDate_ptr->setDemandInformation(lCabinClassPairList,
lYieldDemandPair);
00961         lFoundOnDDate = true;
00962         std::ostringstream oACStr;
00963         for (stdair::AirlineCodeList_T::const_iterator itAC = iAirlineCodeList.
begin();
00964             itAC != iAirlineCodeList.end(); ++itAC) {
00965             if (itAC == iAirlineCodeList.begin()) {
00966                 oACStr << *itAC;
00967             }
00968             else {
00969                 oACStr << "-" << *itAC;
00970             }
00971         }
00972         std::ostringstream oCCStr;
00973         for (stdair::ClassCodeList_T::const_iterator itCC = iClassCodeList.
begin();
00974             itCC != iClassCodeList.end(); ++itCC) {
00975             if (itCC == iClassCodeList.begin()) {
00976                 oCCStr << *itCC;
00977             }
00978             else {
00979                 oCCStr << "-" << *itCC;
00980             }
00981         }
00982         STDAIR_LOG_DEBUG (oACStr.str() << " Classes " << oCCStr.str()
00983             << " Mean " << iMeanValue << " Std Dev " <<
iStdDevValue);
00984         break;
00985     }
00986 }
00987 }
00988 if (!lFoundOnDDate) {
00989     STDAIR_LOG_ERROR ("Cannot find the required multi-segment O&D date: "
00990         << iOrigin << "-" << iDestination << " " <<
iDepartureDate);
00991     assert(false);
00992 }
00993 }
00994
00995 // //////////////////////////////////////
00996 void RMOL_Service::
00997 resetDemandInformation (const stdair::DateTime_T&
iRMEventTime) {
00998     if (_rmolServiceContext == NULL) {
00999         throw stdair::NonInitialisedServiceException ("The Rmol service "
01000             "has not been initialised")

```

```

;
01001     }
01002     assert (_rmolServiceContext != NULL);
01003     RMOL_ServiceContext& lRMOL_ServiceContext = *
_rmolServiceContext;
01004
01005     // Retrieve the bom root
01006     stdair::STDAIR_Service& lSTDAIR_Service =
01007     lRMOL_ServiceContext.getSTDAIR_Service();
01008     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
01009
01010     const stdair::InventoryList_T lInventoryList =
01011     stdair::BomManager::getList<stdair::Inventory> (lBomRoot);
01012     assert (!lInventoryList.empty());
01013     for (stdair::InventoryList_T::const_iterator itInv = lInventoryList.begin()
;
01014         itInv != lInventoryList.end(); ++itInv) {
01015         const stdair::Inventory* lInventory_ptr = *itInv;
01016         assert (lInventory_ptr != NULL);
01017         resetDemandInformation (iRMEventTime, *
lInventory_ptr);
01018     }
01019 }
01020
01021 // //////////////////////////////////////
01022 void RMOL_Service::
01023 resetDemandInformation (const stdair::DateTime_T&
iRMEventTime,
01024                         const stdair::Inventory& iInventory) {
01025
01026     const stdair::FlightDateList_T lFlightDateList =
01027     stdair::BomManager::getList<stdair::FlightDate> (iInventory);
01028     assert (!lFlightDateList.empty());
01029     for (stdair::FlightDateList_T::const_iterator itFD = lFlightDateList.begin(
);
01030         itFD != lFlightDateList.end(); ++itFD) {
01031         const stdair::FlightDate* lFlightDate_ptr = *itFD;
01032         assert (lFlightDate_ptr != NULL);
01033
01034         // Retrieve the date from the RM event
01035         const stdair::Date_T lDate = iRMEventTime.date();
01036
01037         const stdair::Date_T& lDepartureDate = lFlightDate_ptr->getDepartureDate(
);
01038         stdair::DateOffset_T lDateOffset = lDepartureDate - lDate;
01039         stdair::DTD_T lDTD = short (lDateOffset.days());
01040
01041         stdair::DCPLIST_T::const_iterator itDCP =
01042         std::find (stdair::DEFAULT_DCP_LIST.begin(),
stdair::DEFAULT_DCP_LIST.end(), lDTD);
01043         // Check if the demand forecast info corresponding to this flight date
needs to be reset.
01044         if (itDCP != stdair::DEFAULT_DCP_LIST.end()) {
01045             // Check if the flight date holds a list of leg dates.
01046             // If so, find all leg cabin and reset the forecast they are holding.
01047             if (stdair::BomManager::hasList<stdair::LegDate> (*lFlightDate_ptr)) {
01048                 const stdair::LegDateList_T lLegDateList =
01049                 stdair::BomManager::getList<stdair::LegDate> (*lFlightDate_ptr);
01050                 assert (!lLegDateList.empty());
01051                 for (stdair::LegDateList_T::const_iterator itLD = lLegDateList.begin(
);
01052                     itLD != lLegDateList.end(); ++itLD) {
01053                     const stdair::LegDate* lLegDate_ptr = *itLD;
01054                     assert (lLegDate_ptr != NULL);
01055                     const stdair::LegCabinList_T lLegCabinList =
01056                     stdair::BomManager::getList<stdair::LegCabin> (*lLegDate_ptr);
01057                     assert (!lLegCabinList.empty());
01058                     for (stdair::LegCabinList_T::const_iterator itLC = lLegCabinList.
begin();
01059                         itLC != lLegCabinList.end(); ++itLC) {
01060                         stdair::LegCabin* lLegCabin_ptr = *itLC;
01061                         assert (lLegCabin_ptr != NULL);
01062                         lLegCabin_ptr->emptyYieldLevelDemandMap();
01063                     }
01064                 }
01065             }
01066         }
01067     }
01068 }
01069
01070 // //////////////////////////////////////
01071 void RMOL_Service::projectAggregatedDemandOnLegCabins
(const stdair::DateTime_T& iRMEventTime) {
01072
01073     if (_rmolServiceContext == NULL) {
01074         throw stdair::NonInitialisedServiceException ("The Rmol service "
01075                                                         "has not been initialised")

```

```

;
01076     }
01077     assert (_rmolServiceContext != NULL);
01078     RMOL_ServiceContext& lRMOL_ServiceContext = *
_rmolServiceContext;
01079
01080     // Retrieve the bom root
01081     stdair::STDAIR_Service& lSTDAIR_Service =
01082         lRMOL_ServiceContext.getSTDAIR_Service();
01083     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
01084
01085     // Retrieve the date from the RM event
01086     const stdair::Date_T lDate = iRMEventTime.date();
01087
01088     const stdair::InventoryList_T lInventoryList =
01089         stdair::BomManager::getList<stdair::Inventory> (lBomRoot);
01090     assert (!lInventoryList.empty());
01091     for (stdair::InventoryList_T::const_iterator itInv = lInventoryList.begin()
;
01092         itInv != lInventoryList.end(); ++itInv) {
01093         const stdair::Inventory* lInventory_ptr = *itInv;
01094         assert (lInventory_ptr != NULL);
01095         const stdair::OnDDateList_T lOnDDateList =
01096             stdair::BomManager::getList<stdair::OnDDate> (*lInventory_ptr);
01097         assert (!lOnDDateList.empty());
01098         for (stdair::OnDDateList_T::const_iterator itOD = lOnDDateList.begin();
01099             itOD != lOnDDateList.end(); ++itOD) {
01100             stdair::OnDDate* lOnDDate_ptr = *itOD;
01101             assert (lOnDDate_ptr != NULL);
01102
01103             const stdair::Date_T& lDepartureDate = lOnDDate_ptr->getDate();
01104             stdair::DateOffset_T lDateOffset = lDepartureDate - lDate;
01105             stdair::DTD_T lDTD = short (lDateOffset.days());
01106
01107             stdair::DCPList_T::const_iterator itDCP =
01108                 std::find (stdair::DEFAULT_DCP_LIST.begin(),
stdair::DEFAULT_DCP_LIST.end(), lDTD);
01109             // Check if the forecast for this O&D date needs to be projected.
01110             if (itDCP != stdair::DEFAULT_DCP_LIST.end()) {
01111
01112                 // Browse the demand info map.
01113                 const stdair::StringDemandStructMap_T& lStringDemandStructMap =
01114                     lOnDDate_ptr->getDemandInfoMap ();
01115                 for (stdair::StringDemandStructMap_T::const_iterator itStrDS =
lStringDemandStructMap.begin();
01116                     itStrDS != lStringDemandStructMap.end(); ++itStrDS) {
01117                     std::string lCabinClassPath = itStrDS->first;
01118                     const stdair::YieldDemandPair_T& lYieldDemandPair =
01119                         itStrDS->second;
01120                     const stdair::CabinClassPairList_T& lCabinClassPairList =
01121                         lOnDDate_ptr->getCabinClassPairList (lCabinClassPath);
01122                     const unsigned int lNbOfSegments = lOnDDate_ptr->getNbOfSegments();
01123                     // Sanity check
01124                     assert (lCabinClassPairList.size() == lNbOfSegments);
01125
01126                     const stdair::SegmentDateList_T lOnDSegmentDateList =
01127                         stdair::BomManager::getList<stdair::SegmentDate> (*lOnDDate_ptr);
01128                     // Sanity check
01129                     assert (lOnDSegmentDateList.size() == lNbOfSegments);
01130                     stdair::CabinClassPairList_T::const_iterator itCCP =
lCabinClassPairList.begin();
01131                     stdair::SegmentDateList_T::const_iterator itSD =
lOnDSegmentDateList.begin();
01132                     for (; itSD != lOnDSegmentDateList.end(); ++itCCP, ++itSD) {
01133                         const stdair::SegmentDate* lSegmentDate_ptr = *itSD;
01134                         const stdair::SegmentDate* lOperatingSegmentDate_ptr =
01135                             lSegmentDate_ptr->getOperatingSegmentDate ();
01136                         assert (lSegmentDate_ptr != NULL);
01137                         // Only operated legs receive the demand information.
01138                         if (lOperatingSegmentDate_ptr == NULL) {
01139                             const stdair::CabinCode_T lCabinCode = itCCP->first;
01140                             const stdair::ClassCode_T lClassCode = itCCP->second;
01141                             const stdair::SegmentCabin* lSegmentCabin_ptr =
01142                                 stdair::BomManager::getObjectPtr<stdair::SegmentCabin> (*
lSegmentDate_ptr,
01143 lCabinCode);
01144                             assert (lSegmentCabin_ptr != NULL);
01145                             // Retrieve the booking class (level of aggregation of demand).
01146                             // The yield of the class is assigned to all types of demand
01147                             for it.
01148                                 const stdair::BookingClass* lBookingClass_ptr =
01149                                     stdair::BomManager::getObjectPtr<stdair::BookingClass> (*
lSegmentCabin_ptr,
01150 lClassCode);
01151                             assert (lBookingClass_ptr != NULL);

```

```

01151         const stdair::LegCabinList_T lLegCabinList =
01152             stdair::BomManager::getList<stdair::LegCabin>
01153             (*lSegmentCabin_ptr);
01154         assert (!lLegCabinList.empty());
01155         const int lNbOfLegs = lLegCabinList.size();
01156         // Determine the yield (equally distributed over legs).
01157         const stdair::Yield_T& lYield = lBookingClass_ptr->getYield()/
01158             lNbOfLegs;
01159         const stdair::MeanStdDevPair_T& lMeanStdDevPair =
01160             lYieldDemandPair.second;
01161         const stdair::MeanValue_T& lMeanValue = lMeanStdDevPair.first;
01162         const stdair::StdDevValue_T& lStdDevValue = lMeanStdDevPair.
01163             second;
01164         for (stdair::LegCabinList_T::const_iterator itLC =
01165             lLegCabinList.begin();
01166             itLC != lLegCabinList.end(); ++itLC) {
01167             stdair::LegCabin* lLegCabin_ptr = *itLC;
01168             assert (lLegCabin_ptr != NULL);
01169             lLegCabin_ptr->addDemandInformation (lYield, lMeanValue,
01170             lStdDevValue);
01171         }
01172     }
01173 }
01174 }
01175 // //////////////////////////////////////
01176 void RMOL_Service::projectOnDDemandOnLegCabinsUsingYP
01177 (const stdair::DateTime_T& iRMEventTime) {
01178     if (_rmolServiceContext == NULL) {
01179         throw stdair::NonInitialisedServiceException ("The Rmol service "
01180             "has not been initialised")
01181     ;
01182     }
01183     assert (_rmolServiceContext != NULL);
01184     RMOL_ServiceContext& lRMOL_ServiceContext = *
01185         _rmolServiceContext;
01186     // Retrieve the bom root
01187     stdair::STDAIR_Service& lSTDAIR_Service =
01188         lRMOL_ServiceContext.getSTDAIR_Service();
01189     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
01190     // Retrieve the date from the RM event
01191     const stdair::Date_T lDate = iRMEventTime.date();
01192     const stdair::InventoryList_T lInventoryList =
01193         stdair::BomManager::getList<stdair::Inventory> (lBomRoot);
01194     assert (!lInventoryList.empty());
01195     for (stdair::InventoryList_T::const_iterator itInv = lInventoryList.begin()
01196         ;
01197         itInv != lInventoryList.end(); ++itInv) {
01198         const stdair::Inventory* lInventory_ptr = *itInv;
01199         assert (lInventory_ptr != NULL);
01200         const stdair::OnDDateList_T lOnDDateList =
01201             stdair::BomManager::getList<stdair::OnDDate> (*lInventory_ptr);
01202         assert (!lOnDDateList.empty());
01203         for (stdair::OnDDateList_T::const_iterator itOD = lOnDDateList.begin();
01204             itOD != lOnDDateList.end(); ++itOD) {
01205             stdair::OnDDate* lOnDDate_ptr = *itOD;
01206             assert (lOnDDate_ptr != NULL);
01207             const stdair::Date_T lDepartureDate = lOnDDate_ptr->getDate();
01208             stdair::DateOffset_T lDateOffset = lDepartureDate - lDate;
01209             stdair::DTP_T lDTP = short (lDateOffset.days());
01210             stdair::DCPLIST_T::const_iterator itDCP =
01211                 std::find (stdair::DEFAULT_DCP_LIST.begin(),
01212                 stdair::DEFAULT_DCP_LIST.end(), lDTP);
01213             // Check if the forecast for this O&D date needs to be projected.
01214             if (itDCP != stdair::DEFAULT_DCP_LIST.end()) {
01215                 // Browse the demand info map.
01216                 const stdair::StringDemandStructMap_T& lStringDemandStructMap =
01217                     lOnDDate_ptr->getDemandInfoMap ();
01218                 for (stdair::StringDemandStructMap_T::const_iterator itStrDS =
01219                     lStringDemandStructMap.begin();
01220                     itStrDS != lStringDemandStructMap.end(); ++itStrDS) {
01221                     std::string lCabinClassPath = itStrDS->first;
01222                     const stdair::YieldDemandPair_T& lYieldDemandPair =
01223                         itStrDS->second;
01224                     const stdair::CabinClassPairList_T& lCabinClassPairList =
01225                         lOnDDate_ptr->getCabinClassPairList (lCabinClassPath);

```

```

01227         const unsigned int lNbOfSegments = lOnDDate_ptr->getNbOfSegments();
01228         // Sanity check
01229         assert (lCabinClassPairList.size() == lNbOfSegments);
01230
01231         const stdair::SegmentDateList_T lOnDSegmentDateList =
01232             stdair::BomManager::getList<stdair::SegmentDate> (*lOnDDate_ptr);
01233         // Sanity check
01234         assert (lOnDSegmentDateList.size() == lNbOfSegments);
01235         stdair::CabinClassPairList_T::const_iterator itCCP =
01236             lCabinClassPairList.begin();
01237         stdair::SegmentDateList_T::const_iterator itSD =
01238             lOnDSegmentDateList.begin();
01239         for (; itSD != lOnDSegmentDateList.end(); ++itCCP, ++itSD) {
01240             const stdair::SegmentDate* lSegmentDate_ptr = *itSD;
01241             assert (lSegmentDate_ptr != NULL);
01242             const stdair::SegmentDate* lOperatingSegmentDate_ptr =
01243                 lSegmentDate_ptr->getOperatingSegmentDate ();
01244             // Only operated legs receive the demand information.
01245             if (lOperatingSegmentDate_ptr == NULL) {
01246                 const stdair::CabinCode_T lCabinCode = itCCP->first;
01247                 const stdair::ClassCode_T lClassCode = itCCP->second;
01248                 const stdair::SegmentCabin* lSegmentCabin_ptr =
01249                     stdair::BomManager::getObjectPtr<stdair::SegmentCabin> (*
01250                         lSegmentDate_ptr,
01251                         lCabinCode);
01252                 assert (lSegmentCabin_ptr != NULL);
01253                 const stdair::LegCabinList_T lLegCabinList =
01254                     stdair::BomManager::getList<stdair::LegCabin>
01255                         (*lSegmentCabin_ptr);
01256                 assert (!lLegCabinList.empty());
01257                 const int lNbOfLegs = lLegCabinList.size();
01258                 // Determine the yield (equally distributed over segments and
01259                 then legs).
01260                 const stdair::MeanStdDevPair_T& lMeanStdDevPair =
01261                     lYieldDemandPair.second;
01262                 const stdair::Yield_T& lYield = lYieldDemandPair.first/(
01263                     lNbOfLegs*lNbOfSegments);
01264                 const stdair::MeanValue_T& lMeanValue = lMeanStdDevPair.first;
01265                 const stdair::StdDevValue_T& lStdDevValue = lMeanStdDevPair.
01266                     second;
01267                 for (stdair::LegCabinList_T::const_iterator itLC =
01268                     lLegCabinList.begin();
01269                     itLC != lLegCabinList.end(); ++itLC) {
01270                     stdair::LegCabin* lLegCabin_ptr = *itLC;
01271                     assert (lLegCabin_ptr != NULL);
01272                     lLegCabin_ptr->addDemandInformation (lYield, lMeanValue,
01273                         lStdDevValue);
01274                 }
01275             }
01276         }
01277     }
01278 }
01279
01280 ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
01281 void RMOL_Service::optimiseOnD (const
01282     stdair::DateTime_T& iRMEventTime) {
01283     if (_rmolServiceContext == NULL) {
01284         throw stdair::NonInitialisedServiceException ("The Rmol service "
01285             "has not been initialised");
01286     }
01287     assert (_rmolServiceContext != NULL);
01288     RMOL_ServiceContext& lRMOL_ServiceContext = *
01289         _rmolServiceContext;
01290
01291     // Retrieve the bom root
01292     stdair::STDAIR_Service& lSTDAIR_Service =
01293         lRMOL_ServiceContext.getSTDAIR_Service();
01294     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
01295
01296     // Retrieve the date from the RM event
01297     const stdair::Date_T lDate = iRMEventTime.date();
01298
01299     const stdair::InventoryList_T& lInvList =
01300         stdair::BomManager::getList<stdair::Inventory> (lBomRoot);
01301     for (stdair::InventoryList_T::const_iterator itInv = lInvList.begin();
01302         itInv != lInvList.end(); ++itInv) {
01303         stdair::Inventory* lCurrentInv_ptr = *itInv;
01304         assert (lCurrentInv_ptr != NULL);
01305
01306         const stdair::FlightDateList_T& lFlightDateList =
01307             stdair::BomManager::getList<stdair::FlightDate> (*lCurrentInv_ptr);

```



```

01301         for (stdair::FlightDateList_T::const_iterator itFlightDate =
01302             lFlightDateList.begin();
01303             itFlightDate != lFlightDateList.end(); ++itFlightDate) {
01304             stdair::FlightDate* lCurrentFlightDate_ptr = *itFlightDate;
01305             assert (lCurrentFlightDate_ptr != NULL);
01306
01307             const stdair::Date_T& lCurrentDepartureDate = lCurrentFlightDate_ptr->
getDepartureDate();
01308             stdair::DateOffset_T lDateOffset = lCurrentDepartureDate - lDate;
01309             stdair::DTD_T lDTD = short (lDateOffset.days());
01310
01311             stdair::DCPLIST_T::const_iterator itDCP =
01312                 std::find (stdair::DEFAULT_DCP_LIST.begin(),
stdair::DEFAULT_DCP_LIST.end(), lDTD);
01313             // Check if the optimisation is needed.
01314             if (itDCP != stdair::DEFAULT_DCP_LIST.end()) {
01315                 STDAIR_LOG_DEBUG ("Optimisation using O&D forecast: " <<
lCurrentInv_ptr->getAirlineCode()
01316                     << " Departure " << lCurrentDepartureDate << " DTD
" << lDTD);
01317                 Optimiser::optimiseUsingOnDForecast
(*lCurrentFlightDate_ptr);
01318             }
01319         }
01320     }
01321 }
01322
01323 // //////////////////////////////////////
01324 void RMOL_Service::updateBidPrice (const
stdair::DateTime_T& iRMEventTime) {
01325
01326     if (_rmolServiceContext == NULL) {
01327         throw stdair::NonInitialisedServiceException ("The Rmol service "
01328             "has not been initialised")
;
01329     }
01330     assert (_rmolServiceContext != NULL);
01331     RMOL_ServiceContext& lRMOL_ServiceContext = *
_rmolServiceContext;
01332
01333     // Retrieve the bom root
01334     stdair::STDAIR_Service& lSTDAIR_Service =
01335         lRMOL_ServiceContext.getSTDAIR_Service();
01336     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
01337
01338     // Retrieve the date from the RM event
01339     const stdair::Date_T lDate = iRMEventTime.date();
01340
01341     const stdair::InventoryList_T& lInvList =
01342         stdair::BomManager::getList<stdair::Inventory> (lBomRoot);
01343
01344     for (stdair::InventoryList_T::const_iterator itInv = lInvList.begin();
01345         itInv != lInvList.end(); ++itInv) {
01346         stdair::Inventory* lCurrentInv_ptr = *itInv;
01347         assert (lCurrentInv_ptr != NULL);
01348
01349         const stdair::FlightDateList_T& lFlightDateList =
01350             stdair::BomManager::getList<stdair::FlightDate> (*lCurrentInv_ptr);
01351         for (stdair::FlightDateList_T::const_iterator itFlightDate =
01352             lFlightDateList.begin();
01353             itFlightDate != lFlightDateList.end(); ++itFlightDate) {
01354             stdair::FlightDate* lCurrentFlightDate_ptr = *itFlightDate;
01355             assert (lCurrentFlightDate_ptr != NULL);
01356
01357             const stdair::Date_T& lCurrentDepartureDate = lCurrentFlightDate_ptr->
getDepartureDate();
01358             stdair::DateOffset_T lDateOffset = lCurrentDepartureDate - lDate;
01359             stdair::DTD_T lDTD = short (lDateOffset.days());
01360
01361             stdair::DCPLIST_T::const_iterator itDCP =
01362                 std::find (stdair::DEFAULT_DCP_LIST.begin(),
stdair::DEFAULT_DCP_LIST.end(), lDTD);
01363             // Check if the operation is needed.
01364             if (itDCP != stdair::DEFAULT_DCP_LIST.end()) {
01365                 updateBidPrice (*lCurrentFlightDate_ptr, lBomRoot);
01366             }
01367         }
01368     }
01369 }
01370
01371 // //////////////////////////////////////
01372 void RMOL_Service::updateBidPrice (const
stdair::FlightDate& iFlightDate,
01373     stdair::BomRoot& iBomRoot) {
01374     const stdair::SegmentDateList_T& lSegmentDateList =
01375         stdair::BomManager::getList<stdair::SegmentDate> (iFlightDate);
01376     const stdair::AirlineCode_T& lOptAC = iFlightDate.getAirlineCode();

```

```

01377     const std::string lFDKeyStr = iFlightDate.describeKey();
01378
01379     for (stdair::SegmentDateList_T::const_iterator itSegmentDate =
1SegmentDateList.begin();
01380          itSegmentDate != 1SegmentDateList.end(); ++itSegmentDate) {
01381         stdair::SegmentDate* lSegmentDate_ptr = *itSegmentDate;
01382         assert (lSegmentDate_ptr != NULL);
01383         if (stdair::BomManager::hasList<stdair::SegmentDate>(*lSegmentDate_ptr))
01384         {
01385             const stdair::LegDateList_T& lLegDateList =
01386                 stdair::BomManager::getList<stdair::LegDate>(*lSegmentDate_ptr);
01387             // Get the list of marketing carriers segments.
01388             // These are part of maketing partners inventories images held by the
01389             // operating airline.
01390             const stdair::SegmentDateList_T& lMktSegmentDateList =
01391                 stdair::BomManager::getList<stdair::SegmentDate>(*lSegmentDate_ptr);
01392             for (stdair::SegmentDateList_T::const_iterator itMktSD =
1MktSegmentDateList.begin();
01393                  itMktSD != 1MktSegmentDateList.end(); ++itMktSD) {
01394                 // Get the marketing airline code.
01395                 stdair::SegmentDate* lMktSD_ptr = *itMktSD;
01396                 assert (lMktSD_ptr != NULL);
01397                 stdair::FlightDate* lMktFD_ptr =
01398                     stdair::BomManager::getParentPtr<stdair::FlightDate>(*lMktSD_ptr);
01399                 assert (lMktFD_ptr != NULL);
01400                 const stdair::AirlineCode_T& lMktAC = lMktFD_ptr->getAirlineCode();
01401                 // Get the (real) marketer inventory.
01402                 const stdair::Inventory* lMktInv_ptr =
01403                     stdair::BomManager::getObjectPtr<stdair::Inventory>(iBomRoot, lMktAC
);
01404                 assert (lMktInv_ptr != NULL);
01405                 // Get the image of the operating airline inventory held by the
01406                 marketer.
01407                 const stdair::Inventory* lOptInv_ptr =
01408                     stdair::BomManager::getObjectPtr<stdair::Inventory>(*lMktInv_ptr,
1OptAC);
01409                 assert (lOptInv_ptr != NULL);
01410                 // Find the image of the concerned flight date.
01411                 const stdair::FlightDate* lOptFD_ptr =
01412                     stdair::BomManager::getObjectPtr<stdair::FlightDate>(*lOptInv_ptr,
1FDKeyStr);
01413                 assert (lOptFD_ptr != NULL);
01414                 // Browse the list of leg dates in the real operating inventory.
01415                 // Retrieve the image of each leg date.
01416                 for (stdair::LegDateList_T::const_iterator itLD = 1LegDateList.begin(
);
01417                      itLD != 1LegDateList.end(); ++itLD) {
01418                     const stdair::LegDate* lLD_ptr = *itLD;
01419                     assert (lLD_ptr != NULL);
01420                     const std::string lLDKeyStr = lLD_ptr->describeKey();
01421                     stdair::LegDate* lOptLD_ptr =
01422                         stdair::BomManager::getObjectPtr<stdair::LegDate>(*lOptFD_ptr,
1LDKeyStr);
01423                     assert (lOptLD_ptr != NULL);
01424                     const stdair::LegCabinList_T& lLegCabinList_T =
01425                         stdair::BomManager::getList<stdair::LegCabin>(*lLD_ptr);
01426                     // Browse the list of leg cabins in the real operating inventory.
01427                     // Retrieve the image of each leg cabin and update the bid price of
01428                     the real and send it to the image.
01429                     for (stdair::LegCabinList_T::const_iterator itLC = 1LegCabinList_T.
begin();
01430                          itLC != 1LegCabinList_T.end(); ++itLC) {
01431                             stdair::LegCabin* lLC_ptr = *itLC;
01432                             assert (lLC_ptr != NULL);
01433                             const std::string lLCKeyStr = lLC_ptr->describeKey();
01434                             stdair::LegCabin* lOptLC_ptr =
01435                                 stdair::BomManager::getObjectPtr<stdair::LegCabin>(*lOptLD_ptr,
1LCKeyStr);
01436                             assert (lOptLC_ptr != NULL);
01437                             // Update the current bid price of the real leg.
01438                             lLC_ptr->updateCurrentBidPrice();
01439                             // Update the previous bid price (store the current).
01440                             lOptLC_ptr->updatePreviousBidPrice();
01441                             // Update the current bid price.
01442                             lOptLC_ptr->setCurrentBidPrice (lLC_ptr->getCurrentBidPrice());
01443                             STDAIR_LOG_DEBUG ("Update bid price of " << lLC_ptr->getFullerKey
()
01444                                             << " : " << lOptLC_ptr->getCurrentBidPrice()
01445                                             << " Availability pool " << lLC_ptr->
getAvailabilityPool());
01446                             }
01447                         }
01448                     }
01449                 }
01450             }
01451         }
01452     }

```

```

01449
01450 ///////////////////////////////////////////////////////////////////
01451 void RMOL_Service::projectOnDDemandOnLegCabinsUsingDA
(const stdair::DateTime_T& irMEventTime) {
01452
01453     if (_rmolServiceContext == NULL) {
01454         throw stdair::NonInitialisedServiceException ("The Rmol service "
01455             "has not been initialised")
;
01456     }
01457     assert (_rmolServiceContext != NULL);
01458     RMOL_ServiceContext& lRMOL_ServiceContext = *
_rmolServiceContext;
01459
01460     // Retrieve the bom root
01461     stdair::STDAIR_Service& lSTDAIR_Service =
01462         lRMOL_ServiceContext.getSTDAIR_Service();
01463     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
01464
01465     // Retrieve the date from the RM event
01466     const stdair::Date_T lDate = irMEventTime.date();
01467
01468     const stdair::InventoryList_T lInventoryList =
01469         stdair::BomManager::getList<stdair::Inventory> (lBomRoot);
01470     assert (!lInventoryList.empty());
01471     for (stdair::InventoryList_T::const_iterator itInv = lInventoryList.begin()
;
01472         itInv != lInventoryList.end(); ++itInv) {
01473         const stdair::Inventory* lInventory_ptr = *itInv;
01474         assert (lInventory_ptr != NULL);
01475         const stdair::OnDDateList_T lOnDDateList =
01476             stdair::BomManager::getList<stdair::OnDDate> (*lInventory_ptr);
01477         assert (!lOnDDateList.empty());
01478         for (stdair::OnDDateList_T::const_iterator itOD = lOnDDateList.begin();
01479             itOD != lOnDDateList.end(); ++itOD) {
01480             stdair::OnDDate* lOnDDate_ptr = *itOD;
01481             assert (lOnDDate_ptr != NULL);
01482
01483             const stdair::Date_T lDepartureDate = lOnDDate_ptr->getDate();
01484             stdair::DateOffset_T lDateOffset = lDepartureDate - lDate;
01485             stdair::DTD_T lDTD = short (lDateOffset.days());
01486
01487             stdair::DCPLIST_T::const_iterator itDCP =
01488                 std::find (stdair::DEFAULT_DCP_LIST.begin(),
stdair::DEFAULT_DCP_LIST.end(), lDTD);
01489             // Check if the forecast for this O&D date needs to be projected.
01490             if (itDCP != stdair::DEFAULT_DCP_LIST.end()) {
01491
01492                 // Browse the demand info map.
01493                 const stdair::StringDemandStructMap_T& lStringDemandStructMap =
01494                     lOnDDate_ptr->getDemandInfoMap ();
01495                 for (stdair::StringDemandStructMap_T::const_iterator itStrDS =
lStringDemandStructMap.begin();
01496                     itStrDS != lStringDemandStructMap.end(); ++itStrDS) {
01497                     std::string lCabinClassPath = itStrDS->first;
01498                     const stdair::YieldDemandPair_T& lYieldDemandPair = itStrDS->second
;
01499                     const stdair::CabinClassPairList_T& lCabinClassPairList =
01500                         lOnDDate_ptr->getCabinClassPairList (lCabinClassPath);
01501                     const unsigned int lNbOfSegments = lOnDDate_ptr->getNbOfSegments ();
01502                     // Sanity check
01503                     assert (lCabinClassPairList.size() == lNbOfSegments);
01504
01505                     //
01506                     const stdair::SegmentDateList_T lOnDSegmentDateList =
01507                         stdair::BomManager::getList<stdair::SegmentDate> (*lOnDDate_ptr);
01508                     // Sanity check
01509                     assert (lOnDSegmentDateList.size() == lNbOfSegments);
01510                     stdair::CabinClassPairList_T::const_iterator itCCP =
lCabinClassPairList.begin();
01511                     stdair::SegmentDateList_T::const_iterator itSD =
lOnDSegmentDateList.begin();
01512                     // List of bid prices that will be used to easily compute
displacement-adjusted yields.
01513                     std::list<stdair::BidPrice_T> lBidPriceList;
01514                     // The sum of bid prices that will be stored in the list above.
01515                     stdair::BidPrice_T lTotalBidPrice = 0;
01516                     // Retrieve the bid prices
01517                     for (; itSD != lOnDSegmentDateList.end(); ++itCCP, ++itSD) {
01518                         // Get the operating segment cabin (it holds the bid price
information).
01519                         const stdair::SegmentDate* lSegmentDate_ptr = *itSD;
01520                         assert (lSegmentDate_ptr != NULL);
01521                         // Get the operating airline code and check if it is the airline
we are looking for.
01522                         const stdair::SegmentDate* lOperatingSegmentDate_ptr =
01523                             lSegmentDate_ptr->getOperatingSegmentDate ();

```

```

01524         if (lOperatingSegmentDate_ptr != NULL) {
01525             lSegmentDate_ptr = lOperatingSegmentDate_ptr;
01526         }
01527         const stdair::CabinCode_T lCabinCode = itCCP->first;
01528         const stdair::SegmentCabin* lSegmentCabin_ptr =
01529             stdair::BomManager::getObjectPtr<stdair::SegmentCabin> (*
lSegmentDate_ptr,
01530 lCabinCode);
01531         assert (lSegmentCabin_ptr != NULL);
01532         stdair::BidPrice_T lBidPrice = 0;
01533         const stdair::LegCabinList_T lLegCabinList =
01534             stdair::BomManager::getList<stdair::LegCabin>
(*lSegmentCabin_ptr);
01535         for (stdair::LegCabinList_T::const_iterator itLC = lLegCabinList.
begin();
01536             itLC != lLegCabinList.end(); ++itLC) {
01537             const stdair::LegCabin* lLegCabin_ptr = *itLC;
01538             assert (lLegCabin_ptr != NULL);
01539             lBidPrice += lLegCabin_ptr->getCurrentBidPrice();
01540         }
01541         lBidPriceList.push_back (lBidPrice);
01542         lTotalBidPrice += lBidPrice;
01543     }
01544
01545
01546     itCCP = lCabinClassPairList.begin();
01547     itSD = lOnDSegmentDateList.begin();
01548     std::list<stdair::BidPrice_T>::const_iterator itBP = lBidPriceList.
begin();
01549     for (; itSD != lOnDSegmentDateList.end(); ++itCCP, ++itSD, ++itBP)
    {
01550         stdair::BidPrice_T lBidPrice = *itBP;
01551         stdair::BidPrice_T lComplementaryBidPrice = lTotalBidPrice -
lBidPrice;
01552         const stdair::SegmentDate* lSegmentDate_ptr = *itSD;
01553         assert (lSegmentDate_ptr != NULL);
01554         const stdair::SegmentDate* lOperatingSegmentDate_ptr =
lSegmentDate_ptr->getOperatingSegmentDate ();
01555         // Only operated legs receive the demand information.
01556         if (lOperatingSegmentDate_ptr == NULL) {
01557             const stdair::CabinCode_T lCabinCode = itCCP->first;
01558             const stdair::ClassCode_T lClassCode = itCCP->second;
01559             const stdair::SegmentCabin* lSegmentCabin_ptr =
stdair::BomManager::getObjectPtr<stdair::SegmentCabin> (*
lSegmentDate_ptr,
01560 lCabinCode);
01561             assert (lSegmentCabin_ptr != NULL);
01562             const stdair::LegCabinList_T lLegCabinList =
stdair::BomManager::getList<stdair::LegCabin>
(*lSegmentCabin_ptr);
01563             assert (!lLegCabinList.empty());
01564             // Determine the displacement-adjusted yield.
01565             // It is set to 100 (positive small value), if the computed
value is negative.
01566             const stdair::Yield_T& lDAYield =
std::max(100., lYieldDemandPair.first -
lComplementaryBidPrice);
01567
01568             stdair::Yield_T lYield = lDAYield;
01569             // In order to be protected against important variations of
partners' bid price,
01570             // the displacement adjusted yield is not allowed to get out of
a certain range.
01571             // This range is here chosen to be from 80% to 100% of the
(static rule) prorated yield.
01572             /*
01573             const int lNbOfLegs = lLegCabinList.size();
01574             const stdair::Yield_T& lStaticProrationYield =
lDemandStruct.getYield()/(lNbOfLegs*lNbOfSegments);
01575             if (lDAYield < 0.8*lStaticProrationYield){
01576                 lYield = 0.8*lStaticProrationYield;
01577             }
01578             if (lDAYield > lStaticProrationYield) {
01579                 lYield = lStaticProrationYield;
01580             }
01581             */
01582             const stdair::MeanStdDevPair_T& lMeanStdDevPair =
lYieldDemandPair.second;
01583             const stdair::MeanValue_T& lMeanValue = lMeanStdDevPair.first;
01584             const stdair::StdDevValue_T& lStdDevValue = lMeanStdDevPair.
second;
01585             for (stdair::LegCabinList_T::const_iterator itLC =
lLegCabinList.begin();
01586                 itLC != lLegCabinList.end(); ++itLC) {

```

```

01594         stdair::LegCabin* lLegCabin_ptr = *itLC;
01595         assert (lLegCabin_ptr != NULL);
01596         lLegCabin_ptr->addDemandInformation (lYield, lMeanValue,
lStdDevValue);
01597     }
01598 }
01599 }
01600 }
01601 }
01602 }
01603 }
01604 }
01605 }
01606 // //////////////////////////////////////
01607 void RMOL_Service::projectOnDDemandOnLegCabinsUsingDYP
(const stdair::DateTime_T& iRMEventTime) {
01608     if (_rmolServiceContext == NULL) {
01609         throw stdair::NonInitialisedServiceException ("The Rmol service "
01610             "has not been initialised")
;
01612     }
01613     assert (_rmolServiceContext != NULL);
01614     RMOL_ServiceContext& lRMOL_ServiceContext = *
_rmolServiceContext;
01615     // Retrieve the bom root
01616     stdair::STDAIR_Service& lSTDAIR_Service =
lRMOL_ServiceContext.getSTDAIR_Service();
01618     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
01620     const stdair::InventoryList_T lInventoryList =
stdair::BomManager::getList<stdair::Inventory> (lBomRoot);
01622     assert (!lInventoryList.empty());
01623     for (stdair::InventoryList_T::const_iterator itInv = lInventoryList.begin()
;
01625         itInv != lInventoryList.end(); ++itInv) {
01626         const stdair::Inventory* lInventory_ptr = *itInv;
01627         assert (lInventory_ptr != NULL);
01628         projectOnDDemandOnLegCabinsUsingDYP (
iRMEventTime, *lInventory_ptr);
01629     }
01630 }
01631 // //////////////////////////////////////
01632 void RMOL_Service::projectOnDDemandOnLegCabinsUsingDYP
(const stdair::DateTime_T& iRMEventTime,
const
stdair::Inventory& iInventory) {
01635     const stdair::OnDDateList_T lOnDDateList =
stdair::BomManager::getList<stdair::OnDDate> (iInventory);
01637     assert (!lOnDDateList.empty());
01638     for (stdair::OnDDateList_T::const_iterator itOD = lOnDDateList.begin();
itOD != lOnDDateList.end(); ++itOD) {
01640         stdair::OnDDate* lOnDDate_ptr = *itOD;
01642         assert (lOnDDate_ptr != NULL);
01643         // Retrieve the date from the RM event
01644         const stdair::Date_T lDate = iRMEventTime.date();
01645         const stdair::Date_T lDepartureDate = lOnDDate_ptr->getDate();
01648         stdair::DateOffset_T lDateOffset = lDepartureDate - lDate;
01649         stdair::DTD_T lDTD = short (lDateOffset.days());
01650         stdair::DCPLIST_T::const_iterator itDCP =
std::find (stdair::DEFAULT_DCP_LIST.begin(),
stdair::DEFAULT_DCP_LIST.end(), lDTD);
01653         // Check if the forecast for this O&D date needs to be projected.
01654         if (itDCP != stdair::DEFAULT_DCP_LIST.end()) {
01655             // Browse the demand info map.
01657             const stdair::StringDemandStructMap_T& lStringDemandStructMap =
lOnDDate_ptr->getDemandInfoMap ();
01658             for (stdair::StringDemandStructMap_T::const_iterator itStrDS =
lStringDemandStructMap.begin();
itStrDS != lStringDemandStructMap.end(); ++itStrDS) {
01661                 std::string lCabinClassPath = itStrDS->first;
01662                 const stdair::YieldDemandPair_T& lYieldDemandPair = itStrDS->second;
01663                 const stdair::CabinClassPairList_T& lCabinClassPairList =
lOnDDate_ptr->getCabinClassPairList (lCabinClassPath);
01664                 const unsigned int lNbOfSegments = lOnDDate_ptr->getNbOfSegments();
01665                 // Sanity check
01666                 assert (lCabinClassPairList.size() == lNbOfSegments);
01667             //
01668             const stdair::SegmentDateList_T lOnDSegmentDateList =

```

```

01671         stdair::BomManager::getList<stdair::SegmentDate> (*lOnDDate_ptr);
01672         // Sanity check
01673         assert (lOnDSegmentDateList.size() == lNbOfSegments);
01674         stdair::CabinClassPairList_T::const_iterator itCCP =
01675         lCabinClassPairList.begin();
01676         stdair::SegmentDateList_T::const_iterator itSD = lOnDSegmentDateList.
01677         begin();
01678         // The sum of bid prices of all cabins.
01679         stdair::BidPrice_T lTotalBidPrice = 0;
01680         for (; itSD != lOnDSegmentDateList.end(); ++itCCP, ++itSD) {
01681             // Get the operating segment cabin (it holds the bid price
01682             information).
01683             const stdair::SegmentDate* lSegmentDate_ptr = *itSD;
01684             assert (lSegmentDate_ptr != NULL);
01685             // Get the operating airline code and check if it is the airline we
01686             are looking for.
01687             const stdair::SegmentDate* lOperatingSegmentDate_ptr =
01688             lSegmentDate_ptr->getOperatingSegmentDate ();
01689             if (lOperatingSegmentDate_ptr != NULL) {
01690                 lSegmentDate_ptr = lOperatingSegmentDate_ptr;
01691             }
01692             const stdair::CabinCode_T lCabinCode = itCCP->first;
01693             const stdair::SegmentCabin* lSegmentCabin_ptr =
01694             stdair::BomManager::getObjectPtr<stdair::SegmentCabin> (*
01695             lSegmentDate_ptr,
01696             lCabinCode);
01697             assert (lSegmentCabin_ptr != NULL);
01698             const stdair::LegCabinList_T lLegCabinList =
01699             stdair::BomManager::getList<stdair::LegCabin>(*lSegmentCabin_ptr)
01700             ;
01701             for (stdair::LegCabinList_T::const_iterator itLC = lLegCabinList.
01702             begin();
01703                 itLC != lLegCabinList.end(); ++itLC) {
01704                 const stdair::LegCabin* lLegCabin_ptr = *itLC;
01705                 assert (lLegCabin_ptr != NULL);
01706                 lTotalBidPrice += lLegCabin_ptr->getCurrentBidPrice();
01707             }
01708         }
01709         itCCP = lCabinClassPairList.begin();
01710         itSD = lOnDSegmentDateList.begin();
01711         for (; itSD != lOnDSegmentDateList.end(); ++itCCP, ++itSD) {
01712             const stdair::SegmentDate* lSegmentDate_ptr = *itSD;
01713             assert (lSegmentDate_ptr != NULL);
01714             const stdair::SegmentDate* lOperatingSegmentDate_ptr =
01715             lSegmentDate_ptr->getOperatingSegmentDate ();
01716             // Only operated legs receive the demand information.
01717             if (lOperatingSegmentDate_ptr == NULL) {
01718                 const stdair::CabinCode_T lCabinCode = itCCP->first;
01719                 const stdair::ClassCode_T lClassCode = itCCP->second;
01720                 const stdair::SegmentCabin* lSegmentCabin_ptr =
01721                 stdair::BomManager::getObjectPtr<stdair::SegmentCabin> (*
01722                 lSegmentDate_ptr,
01723                 lCabinCode);
01724                 assert (lSegmentCabin_ptr != NULL);
01725                 const stdair::LegCabinList_T lLegCabinList =
01726                 stdair::BomManager::getList<stdair::LegCabin>
01727                 (*lSegmentCabin_ptr);
01728                 assert (!lLegCabinList.empty());
01729                 const stdair::Yield_T& lYield = lYieldDemandPair.first;
01730
01731                 const stdair::MeanStdDevPair_T& lMeanStdDevPair =
01732                 lYieldDemandPair.second;
01733                 const stdair::MeanValue_T& lMeanValue = lMeanStdDevPair.first;
01734                 const stdair::StdDevValue_T& lStdDevValue = lMeanStdDevPair.
01735                 second;
01736                 for (stdair::LegCabinList_T::const_iterator itLC = lLegCabinList.
01737                 begin();
01738                     itLC != lLegCabinList.end(); ++itLC) {
01739                     stdair::LegCabin* lLegCabin_ptr = *itLC;
01740                     assert (lLegCabin_ptr != NULL);
01741                     const stdair::BidPrice_T& lBidPrice = lLegCabin_ptr->
01742                     getCurrentBidPrice();
01743                     const stdair::RealNumber_T lDynamicYieldProrationFactor =
01744                     lBidPrice / lTotalBidPrice;
01745                     const stdair::Yield_T lProratedYield =
01746                     lDynamicYieldProrationFactor*lYield;
01747                     lLegCabin_ptr->addDemandInformation (lProratedYield, lMeanValue
01748                     , lStdDevValue);
01749                 }
01750                 // STDAIR_LOG_DEBUG ("Adding demand information to leg-cabin "
01751                 << lLegCabin_ptr->getFullerKey()
01752                 //
01753                 << " Total yield " << lYield << "
01754                 Proration factor "

```

```

01738          //          << lDynamicYieldProrationFactor << "
Prorated yield " << lProratedYield
01739          //          << " Mean demand " << lMeanValue << "
StdDev " << lStdDevValue);
01740      }
01741  }
01742  }
01743  }
01744  }
01745  }
01746  }
01747
01748  // //////////////////////////////////////
01749  void RMOL_Service::optimiseOnDUsingRMCooperation
(const stdair::DateTime_T& iRMEventTime) {
01750
01751      if (_rmolServiceContext == NULL) {
01752          throw stdair::NonInitialisedServiceException ("The Rmol service "
01753              "has not been initialised")
;
01754      }
01755      assert (_rmolServiceContext != NULL);
01756      RMOL_ServiceContext& lRMOL_ServiceContext = *
_rmolServiceContext;
01757
01758      // Retrieve the bom root
01759      stdair::STDAIR_Service& lSTDAIR_Service =
01760          lRMOL_ServiceContext.getSTDAIR_Service();
01761      stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
01762
01763      // Retrieve the date from the RM event
01764      const stdair::Date_T lDate = iRMEventTime.date();
01765
01766      // Browse the list of inventories and optimise within each one
independently.
01767      const stdair::InventoryList_T& lInvList =
01768          stdair::BomManager::getList<stdair::Inventory> (lBomRoot);
01769      for (stdair::InventoryList_T::const_iterator itInv = lInvList.begin();
01770          itInv != lInvList.end(); ++itInv) {
01771          stdair::Inventory* lCurrentInv_ptr = *itInv;
01772          assert (lCurrentInv_ptr != NULL);
01773
01774          double lMaxBPVariation = 1.0;
01775          short lIterationCounter = 0;
01776          // Iterate until the variation is under the wanted level or the maximal
number of iterations is reached.
01777          while (lMaxBPVariation > 0.01 && lIterationCounter < 10) {
01778              lMaxBPVariation = 0.0;
01779              lIterationCounter++;
01780              const stdair::FlightDateList_T& lFlightDateList =
01781                  stdair::BomManager::getList<stdair::FlightDate> (*lCurrentInv_ptr);
01782              for (stdair::FlightDateList_T::const_iterator itFlightDate =
01783                  lFlightDateList.begin();
01784                  itFlightDate != lFlightDateList.end(); ++itFlightDate) {
01785                  stdair::FlightDate* lCurrentFlightDate_ptr = *itFlightDate;
01786                  assert (lCurrentFlightDate_ptr != NULL);
01787
01788                  const stdair::Date_T& lCurrentDepartureDate = lCurrentFlightDate_ptr
->getDepartureDate();
01789                  stdair::DateOffset_T lDateOffset = lCurrentDepartureDate - lDate;
01790                  stdair::DTD_T lDTD = short (lDateOffset.days());
01791
01792                  stdair::DCPLList_T::const_iterator itDCP =
01793                      std::find (stdair::DEFAULT_DCP_LIST.begin()
, stdair::DEFAULT_DCP_LIST.end(), lDTD);
01794                  // Check if the optimisation is needed.
01795                  if (itDCP != stdair::DEFAULT_DCP_LIST.end())
{
01796                      const double lBPVariation = Optimiser::optimiseUsingOnDForecast
(*lCurrentFlightDate_ptr);
01797                      lMaxBPVariation = std::max(lMaxBPVariation, lBPVariation);
01798                  }
01799              }
01800              // Update the prorated yields for the current inventory.
01801              resetDemandInformation (iRMEventTime, *
lCurrentInv_ptr);
01802              projectOnDDemandOnLegCabinsUsingDYP
(iRMEventTime, *lCurrentInv_ptr);
01803          }
01804      }
01805  }
01806
01807  // //////////////////////////////////////
01808
01809  void RMOL_Service::optimiseOnDUsingAdvancedRMCooperation
(const stdair::DateTime_T& iRMEventTime) {
01810

```

```

01811     if (_rmolServiceContext == NULL) {
01812         throw stdair::NonInitialisedServiceException ("The Rmol service "
01813             "has not been initialised")
    ;
01814     }
01815     assert (_rmolServiceContext != NULL);
01816     RMOL_ServiceContext& lRMOL_ServiceContext = *
        _rmolServiceContext;
01817
01818     // Retrieve the bom root
01819     stdair::STDAIR_Service& lSTDAIR_Service =
01820         lRMOL_ServiceContext.getSTDAIR_Service();
01821     stdair::BomRoot& lBomRoot = lSTDAIR_Service.getBomRoot();
01822
01823     // Retrieve the date from the RM event
01824     const stdair::Date_T lDate = iRMEventTime.date();
01825
01826     double lMaxBPVariation = 1.0;
01827     short lIterationCounter = 0;
01828     // Iterate until the variation is under the wanted level or the maximal
    number of iterations is reached.
01829     // Every iteration corresponds to the optimisation of the whole network.
    Bid prices are communicated
    // between partners at the end of each iteration.
01831     while (lMaxBPVariation > 0.01 && lIterationCounter < 50) {
01832         lMaxBPVariation = 0.0;
01833         lIterationCounter++;
01834
01835         const stdair::InventoryList_T& lInvList =
01836             stdair::BomManager::getList<stdair::Inventory> (lBomRoot);
01837         for (stdair::InventoryList_T::const_iterator itInv = lInvList.begin();
01838             itInv != lInvList.end(); ++itInv) {
01839             stdair::Inventory* lCurrentInv_ptr = *itInv;
01840             assert (lCurrentInv_ptr != NULL);
01841             const stdair::FlightDateList_T& lFlightDateList =
01842                 stdair::BomManager::getList<stdair::FlightDate> (*lCurrentInv_ptr);
01843             for (stdair::FlightDateList_T::const_iterator itFlightDate =
01844                 lFlightDateList.begin();
01845                 itFlightDate != lFlightDateList.end(); ++itFlightDate) {
01846                 stdair::FlightDate* lCurrentFlightDate_ptr = *itFlightDate;
01847                 assert (lCurrentFlightDate_ptr != NULL);
01848
01849                 const stdair::Date_T& lCurrentDepartureDate = lCurrentFlightDate_ptr
->getDepartureDate();
01850                 stdair::DateOffset_T lDateOffset = lCurrentDepartureDate - lDate;
01851                 stdair::DTD_T lDTD = short (lDateOffset.days());
01852
01853                 stdair::DCPLIST_T::const_iterator itDCP =
01854                     std::find (stdair::DEFAULT_DCP_LIST.begin()
, stdair::DEFAULT_DCP_LIST.end(), lDTD);
01855                 if (itDCP != stdair::DEFAULT_DCP_LIST.end())
    {
01856                     const double lBPVariation = Optimiser::optimiseUsingOnDForecast
(*lCurrentFlightDate_ptr);
01857                     lMaxBPVariation = std::max(lMaxBPVariation, lBPVariation);
01858                 }
01859             }
01860         }
01861         // At the end of each iteration, communicate bid prices and compute
    displacement adjusted yields.
01862         updateBidPrice (iRMEventTime);
01863         resetDemandInformation (iRMEventTime);
01864         projectOnDDemandOnLegCabinsUsingDYP (
iRMEventTime);
01865     }
01866 }
01867
01868 }

```

26.97 rmol/service/RMOL_ServiceContext.cpp File Reference

```

#include <cassert>
#include <sstream>
#include <stdair/STDAIR_Service.hpp>
#include <rmol/basic/BasConst_RMOL_Service.hpp>
#include <rmol/service/RMOL_ServiceContext.hpp>

```


Namespaces

- namespace [RMOL](#)

26.98 RMOL_ServiceContext.cpp

```

00001 // //////////////////////////////////////
00002 // Import section
00003 // //////////////////////////////////////
00004 // STL
00005 #include <cassert>
00006 #include <sstream>
00007 // StdAir
00008 #include <stdair/STDAIR_Service.hpp>
00009 // RMOL
00010 #include <rmol/basic/BasConst_RMOL_Service.hpp>
00011 >
00012 #include <rmol/service/RMOL_ServiceContext.hpp>
00013 >
00012 namespace RMOL {
00013
00014 // //////////////////////////////////////
00015 RMOL_ServiceContext::RMOL_ServiceContext() : _ownStdairService (false) {
00016 }
00017
00018 // //////////////////////////////////////
00019 RMOL_ServiceContext::RMOL_ServiceContext (const RMOL_ServiceContext&) {
00020     assert (false);
00021 }
00022
00023 // //////////////////////////////////////
00024 RMOL_ServiceContext::~RMOL_ServiceContext() {
00025 }
00026
00027 // //////////////////////////////////////
00028 stdair::STDAIR_Service& RMOL_ServiceContext::getSTDAIR_Service() const {
00029     assert (_stdairService != NULL);
00030     return *_stdairService;
00031 }
00032
00033 // //////////////////////////////////////
00034 const std::string RMOL_ServiceContext::shortDisplay() const {
00035     std::ostringstream oStr;
00036     oStr << "RMOL_ServiceContext -- Owns StdAir service: " << _ownStdairService;
00037
00038     return oStr.str();
00039 }
00040
00041 // //////////////////////////////////////
00042 const std::string RMOL_ServiceContext::display() const {
00043     std::ostringstream oStr;
00044     oStr << shortDisplay();
00045     return oStr.str();
00046 }
00047
00048 // //////////////////////////////////////
00049 const std::string RMOL_ServiceContext::describe() const {
00050     return shortDisplay();
00051 }
00052
00053 // //////////////////////////////////////
00054 void RMOL_ServiceContext::reset() {
00055     if (_ownStdairService == true) {
00056         _stdairService.reset();
00057     }
00058 }
00059
00060 }

```

26.99 rmol/service/RMOL_ServiceContext.hpp File Reference

```
#include <string>
#include <stdair/stdair_basic_types.hpp>
#include <stdair/stdair_inventory_types.hpp>
#include <stdair/stdair_maths_types.hpp>
#include <stdair/stdair_service_types.hpp>
#include <stdair/service/ServiceAbstract.hpp>
#include <rmol/RMOL_Types.hpp>
```

Classes

- class [RMOL::RMOL_ServiceContext](#)
Inner class holding the context for the [RMOL](#) Service object.

Namespaces

- namespace [stdair](#)
- namespace [RMOL](#)

26.100 RMOL_ServiceContext.hpp

```
00001 #ifndef __RMOL_SVC_RMOL_SERVICE_CONTEXT_HPP
00002 #define __RMOL_SVC_RMOL_SERVICE_CONTEXT_HPP
00003
00004 // //////////////////////////////////////
00005 // Import section
00006 // //////////////////////////////////////
00007 // STL
00008 #include <string>
00009 // StdAir
00010 #include <stdair/stdair_basic_types.hpp>
00011 #include <stdair/stdair_inventory_types.hpp>
00012 #include <stdair/stdair_maths_types.hpp>
00013 #include <stdair/stdair_service_types.hpp>
00014 #include <stdair/service/ServiceAbstract.hpp>
00015 // RMOL
00016 #include <rmol/RMOL_Types.hpp>
00017
00019 namespace stdair {
00020     class STDAIR_Service;
00021     class LegCabin;
00022 }
00023
00024 namespace RMOL {
00025
00029     class RMOL_ServiceContext : public stdair::ServiceAbstract
00030     {
00035         friend class RMOL_Service;
00036         friend class FacRmolServiceContext;
00037
00038     private:
00039         // ////////////////////////////////// Getters //////////////////////////////////
00043         stdair::STDAIR_ServicePtr_T getSTDAIR_ServicePtr() const {
00044             return _stdairService;
00045         }
00046
00050         stdair::STDAIR_Service& getSTDAIR_Service() const;
00051
00055         const bool getOwnStdairServiceFlag() const {
00056             return _ownStdairService;
00057         }
00058
00059
00060     private:
00061         // ////////////////////////////////// Setters //////////////////////////////////
00065         void setSTDAIR_Service (stdair::STDAIR_ServicePtr_T ioSTDAIR_ServicePtr,
00066                                 const bool iOwnStdairService) {
00067             _stdairService = ioSTDAIR_ServicePtr;
00068             _ownStdairService = iOwnStdairService;
00069         }
00070     }
```

```

00070
00074     void reset();
00075
00076
00077 private:
00078     // /////////// Display Methods ///////////
00082     const std::string shortDisplay() const;
00083
00087     const std::string display() const;
00088
00092     const std::string describe() const;
00093
00094
00095 private:
00096     // /////////// Construction / initialisation ///////////
00100     RMOL_ServiceContext();
00104     RMOL_ServiceContext (const RMOL_ServiceContext&);
00105
00109     ~RMOL_ServiceContext();
00110
00111
00112 private:
00113     // /////////// Children ///////////
00117     stdair::STDAIR_ServicePtr_T _stdairService;
00118
00122     bool _ownStdairService;
00123 };
00124
00125 }
00126 #endif // __RMOL_SVC_RMOL_SERVICE_CONTEXT_HPP

```

26.101 test/rmol/bomsforforecaster.cpp File Reference

26.102 bomsforforecaster.cpp

```

00001
00005 // //////////////////////////////////////
00006 // Import section
00007 // //////////////////////////////////////
00008 // STL
00009 #include <cassert>
00010 #include <limits>
00011 #include <sstream>
00012 #include <fstream>
00013 #include <string>
00014 // Boost Unit Test Framework (UTF)
00015 #define BOOST_TEST_DYN_LINK
00016 #define BOOST_TEST_MAIN
00017 #define BOOST_TEST_MODULE OptimiseTestSuite
00018 #include <boost/test/unit_test.hpp>
00019 // StdAir
00020 #include <stdair/basic/BasLogParams.hpp>
00021 #include <stdair/basic/BasDBParams.hpp>
00022 #include <stdair/service/Logger.hpp>
00023 // RMOL
00024 #include <rmol/RMOL_Service.hpp>
00025 #include <rmol/config/rmol-paths.hpp>
00026
00027 namespace boost_utf = boost::unit_test;
00028
00029 // (Boost) Unit Test XML Report
00030 std::ofstream utfReportStream ("bomsforforecaster_utfresults.xml");
00031
00035 struct UnitTestConfig {
00037     UnitTestConfig() {
00038         boost_utf::unit_test_log.set_stream (utfReportStream);
00039         boost_utf::unit_test_log.set_format (boost_utf::XML);
00040         boost_utf::unit_test_log.set_threshold_level (boost_utf::log_test_units);
00041         //boost_utf::unit_test_log.set_threshold_level
00042         (boost_utf::log_successful_tests);
00043     }
00044
00045     ~UnitTestConfig() {
00046     }
00047 };
00048
00049 namespace RMOL {
00050
00052     struct BookingClassData {
00053
00054         // Attributes
00055         double _bookingCount;
00056         double _fare;

```

```

00057     double _sellupFactor;
00058     bool _censorshipFlag;
00059
00060     // Constructor
00061     BookingClassData (const double iBookingCount, const double iFare,
00062                      const double iSellupFactor, const bool iCensorshipFlag)
00063         : _bookingCount(iBookingCount), _fare(iFare),
00064           _sellupFactor(iSellupFactor), _censorshipFlag(iCensorshipFlag) {
00065     }
00066
00067     // Getters
00068     double getFare () const {
00069         return _fare;
00070     }
00071
00072     bool getCensorshipFlag () const {
00073         return _censorshipFlag;
00074     }
00075
00076     // Display
00077     std::string toString() const {
00078         std::ostringstream ostr;
00079         ostr << std::endl
00080             << "[Booking class data information]" << std::endl
00081             << "Booking counter: " << _bookingCount << std::endl
00082             << "Fare: " << _fare << std::endl
00083             << "Sell-up Factor: " << _sellupFactor << std::endl
00084             << "censorshipFlag: " << _censorshipFlag << std::endl;
00085         return ostr.str();
00086     }
00087 };
00088
00089 struct BookingClassDataSet {
00090
00091     typedef std::vector<BookingClassData*> BookingClassDataList_T;
00092
00093     // Attributes
00094     int _numberOfClass;
00095     double _minimumFare;
00096     bool _censorshipFlag; // true if any of the classes is censored
00097     BookingClassDataList_T _bookingClassDataList;
00098
00099     // Constructor
00100     BookingClassDataSet ()
00101         : _numberOfClass(0), _minimumFare(0),
00102           _censorshipFlag(false) {
00103     }
00104
00105     // Add BookingClassData
00106     void addBookingClassData (BookingClassData& ioBookingClassData) {
00107         _bookingClassDataList.push_back (&ioBookingClassData);
00108     }
00109
00110     // Getters
00111     unsigned int getNumberOfClass () const {
00112         return _bookingClassDataList.size();
00113     }
00114
00115     double getMinimumFare () const {
00116         return _minimumFare;
00117     }
00118
00119     bool getCensorshipFlag () const {
00120         return _censorshipFlag;
00121     }
00122
00123     // Setters
00124     void setMinimumFare (const double iMinFare) {
00125         _minimumFare = iMinFare;
00126     }
00127
00128     void setCensorshipFlag (const bool iCensorshipFlag) {
00129         _censorshipFlag = iCensorshipFlag;
00130     }
00131
00132     // compute minimum fare
00133     void updateMinimumFare() {
00134         double minFare = std::numeric_limits<double>::max();
00135         BookingClassDataList_T::iterator itBookingClassDataList;
00136         for (itBookingClassDataList = _bookingClassDataList.begin();
00137             itBookingClassDataList != _bookingClassDataList.end();
00138             ++itBookingClassDataList) {
00139             BookingClassData* lBookingClassData = *itBookingClassDataList;
00140             assert (lBookingClassData != NULL);
00141             const double lFare = lBookingClassData->getFare();

```

```

00145         if (lFare < minFare) {
00146             minFare = lFare;
00147         }
00148     }
00149     //
00150     setMinimumFare(minFare);
00151 }
00152
00153 // compute censorship flag for the data set
00154 void updateCensorshipFlag () {
00155     bool censorshipFlag = false;
00156     BookingClassDataList_T::iterator itBookingClassDataList;
00157     for (itBookingClassDataList = _bookingClassDataList.begin();
00158         itBookingClassDataList != _bookingClassDataList.end();
00159         ++itBookingClassDataList) {
00160         BookingClassData* lBookingClassData = *itBookingClassDataList;
00161         assert (lBookingClassData != NULL);
00162
00163         const bool lCensorshipFlagOfAClass =
00164             lBookingClassData->getCensorshipFlag();
00165         if (lCensorshipFlagOfAClass) {
00166             censorshipFlag = true;
00167             break;
00168         }
00169     }
00170     //
00171     setCensorshipFlag(censorshipFlag);
00172 }
00173
00174 // Display
00175 std::string toString() const {
00176     std::ostringstream ostr;
00177     ostr << std::endl
00178         << "[Booking class data set information]" << std::endl
00179         << "Number of classes: " << _numberOfClass << std::endl
00180         << "Minimum fare: " << _minimumFare << std::endl
00181         << "The data of the class set are sensed: " << _censorshipFlag
00182         << std::endl;
00183     return ostr.str();
00184 }
00185
00186 };
00187
00188 // /*----- BOM : Q-Forecaster ----- */
00189 // struct QForecaster {
00190
00191 //     // Function focused BOM
00192
00193 //     // 1. calculate sell up probability for Q-eq
00194
00195 //     // 2. calculate Q-Equivalent Booking
00196 //     double calculateQEqBooking (BookingClassDataSet& iBookingClassDataSet) {
00197 //         double lQEqBooking = 0.0;
00198 //         double lMinFare = iBookingClassDataSet.getMinimumFare();
00199
00200 //         return lQEqBooking;
00201 //     }
00202 //
00203 //     /* Calculate Q-equivalent demand
00204 //     [<- performed by unconstrainer if necessary (Using ExpMax BOM)]
00205 //     */
00206 //
00207 //
00208 //     // 3. Partition to each class
00209 //
00210 //
00211 //
00212 // };
00213 // };
00214 }
00215
00216 // ////////////////////////////////// Main: Unit Test Suite //////////////////////////////////
00217
00218 // Set the UTF configuration (re-direct the output to a specific file)
00220 BOOST_GLOBAL_FIXTURE (UnitTestFixture);
00221
00225 BOOST_AUTO_TEST_SUITE (master_test_suite)
00226
00227
00230 BOOST_AUTO_TEST_CASE (rmol_forecaster) {
00231
00232     // Output log File
00233     std::string lLogFilename ("bomsforforecaster.log");
00234     std::ofstream logOutputFile;
00235
00236     // Open and clean the log outputfile

```

```

00237 logOutputFile.open (lLogFilename.c_str());
00238 logOutputFile.clear();
00239
00240 // Initialise the RMOL service
00241 const stdair::BasLogParams lLogParams (stdair::LOG::DEBUG, logOutputFile);
00242
00243 // Initialise the RMOL service
00244 RMOL::RMOL_Service rmolService (lLogParams);
00245
00246 // Build a sample BOM tree
00247 rmolService.buildSampleBom();
00248
00249 // Register BCDataset
00250 RMOL::BookingClassDataSet lBookingClassDataSet;
00251
00252 // Register BookingClassData
00253 RMOL::BookingClassData QClassData (10, 100, 1, false);
00254 RMOL::BookingClassData MClassData (5, 150, 0.8, true);
00255 RMOL::BookingClassData BClassData (0, 200, 0.6, false);
00256 RMOL::BookingClassData YClassData (0, 300, 0.3, false);
00257
00258 // Display
00259 STDAIR_LOG_DEBUG (QClassData.toString());
00260 STDAIR_LOG_DEBUG (MClassData.toString());
00261 STDAIR_LOG_DEBUG (BClassData.toString());
00262 STDAIR_LOG_DEBUG (YClassData.toString());
00263
00264 // Add BookingClassData into the BCDataset
00265 lBookingClassDataSet.addBookingClassData (QClassData);
00266 lBookingClassDataSet.addBookingClassData (MClassData);
00267 lBookingClassDataSet.addBookingClassData (BClassData);
00268 lBookingClassDataSet.addBookingClassData (YClassData);
00269
00270 // DEBUG
00271 STDAIR_LOG_DEBUG (lBookingClassDataSet.toString());
00272
00273 // Number of classes
00274 const unsigned int lNoOfClass = lBookingClassDataSet.getNumberOfClass();
00275
00276 // DEBUG
00277 STDAIR_LOG_DEBUG ("Number of Classes: " << lNoOfClass);
00278
00279 // Minimum fare
00280 BOOST_CHECK_NO_THROW (lBookingClassDataSet.updateMinimumFare());
00281 const double lMinFare = lBookingClassDataSet.getMinimumFare();
00282
00283 // DEBUG
00284 STDAIR_LOG_DEBUG ("Minimum fare: " << lMinFare);
00285
00286 // Censorship flag
00287 BOOST_CHECK_NO_THROW (lBookingClassDataSet.updateCensorshipFlag());
00288 const bool lCensorshipFlag = lBookingClassDataSet.getCensorshipFlag();
00289
00290 // DEBUG
00291 STDAIR_LOG_DEBUG ("Censorship Flag: " << lCensorshipFlag);
00292
00293 // Close the log output file
00294 logOutputFile.close();
00295 }
00296
00297 // End the test suite
00298 BOOST_AUTO_TEST_SUITE_END()
00299
00300

```

26.103 test/rmol/ForecasterTestSuite.cpp File Reference

26.104 ForecasterTestSuite.cpp

```

00001
00005 // //////////////////////////////////////
00006 // Import section
00007 // //////////////////////////////////////
00008 // STL
00009 #include <sstream>
00010 #include <fstream>
00011 #include <string>
00012 #include <vector>
00013 #include <cmath>
00014 // Boost Unit Test Framework (UTF)
00015 #define BOOST_TEST_DYN_LINK
00016 #define BOOST_TEST_MAIN
00017 #define BOOST_TEST_MODULE ForecasterTestSuite

```

```

00018 #include <boost/test/unit_test.hpp>
00019 // StdAir
00020 #include <stdair/basic/BasLogParams.hpp>
00021 #include <stdair/basic/BasDBParams.hpp>
00022 #include <stdair/basic/BasFileMgr.hpp>
00023 #include <stdair/service/Logger.hpp>
00024 // RMOL
00025 #include <rmol/RMOL_Service.hpp>
00026
00027 namespace boost_utf = boost::unit_test;
00028
00029 // (Boost) Unit Test XML Report
00030 std::ofstream utfReportStream ("ForecasterTestSuite_utfresults.xml");
00031
00032 struct UnitTestConfig {
00033     UnitTestConfig() {
00034         boost_utf::unit_test_log.set_stream (utfReportStream);
00035         boost_utf::unit_test_log.set_format (boost_utf::XML);
00036         boost_utf::unit_test_log.set_threshold_level (boost_utf::log_test_units);
00037         //boost_utf::unit_test_log.set_threshold_level
00038         (boost_utf::log_successful_tests);
00039     }
00040
00041     ~UnitTestConfig() {
00042     }
00043 };
00044
00045 // Main: Unit Test Suite
00046 // Set the UTF configuration (re-direct the output to a specific file)
00047 BOOST_GLOBAL_FIXTURE (UnitTestConfig);
00048 BOOST_AUTO_TEST_SUITE (master_test_suite)
00049
00050 BOOST_AUTO_TEST_CASE (rmol_forecaster_q_forecasting) {
00051     const bool lTestFlag = true; //testForecasterHelper(0);
00052     BOOST_CHECK_EQUAL (lTestFlag, true);
00053     BOOST_CHECK_MESSAGE (lTestFlag == true,
00054         "The test has failed. Please see the log file for "
00055         "<< \"more details\");
00056 }
00057
00058 // End the test suite
00059 BOOST_AUTO_TEST_SUITE_END()
00060
00061
00062

```

26.105 test/rmol/ForecasterTestSuite.hpp File Reference

```

#include <sstream>
#include <cppunit/extensions/HelperMacros.h>

```

Classes

- class [ForecasterTestSuite](#)

Functions

- [CPPUNIT_TEST_SUITE_REGISTRATION](#) ([ForecasterTestSuite](#))

26.105.1 Function Documentation

26.105.1.1 CPPUNIT_TEST_SUITE_REGISTRATION (ForecasterTestSuite)

26.106 ForecasterTestSuite.hpp

```

00001 // STL
00002 #include <sstream>
00003 // CPPUNIT

```

```

00004 #include <cppunit/extensions/HelperMacros.h>
00005
00006 class ForecasterTestSuite : public CppUnit::TestFixture {
00007     CPPUNIT_TEST_SUITE (ForecasterTestSuite);
00008     CPPUNIT_TEST (testQForecaster);
00009     CPPUNIT_TEST_SUITE_END ();
00010 public:
00011
00013     void testQForecaster();
00014
00016     ForecasterTestSuite ();
00017
00018 protected:
00019     std::stringstream _describeKey;
00020 };
00021
00022 CPPUNIT_TEST_SUITE_REGISTRATION (
    ForecasterTestSuite);

```

26.107 test/rmol/OptimiseTestSuite.cpp File Reference

26.108 OptimiseTestSuite.cpp

```

00001
00005 // ////////////////////////////////////////
00006 // Import section
00007 // ////////////////////////////////////////
00008 // STL
00009 #include <sstream>
00010 #include <fstream>
00011 #include <string>
00012 // Boost Unit Test Framework (UTF)
00013 #define BOOST_TEST_DYN_LINK
00014 #define BOOST_TEST_MAIN
00015 #define BOOST_TEST_MODULE OptimiseTestSuite
00016 #include <boost/test/unit_test.hpp>
00017 // StdAir
00018 #include <stdair/basic/BasLogParams.hpp>
00019 #include <stdair/basic/BasDBParams.hpp>
00020 #include <stdair/basic/BasFileMgr.hpp>
00021 #include <stdair/service/Logger.hpp>
00022 // RMOL
00023 #include <rmol/RMOL_Service.hpp>
00024 #include <rmol/config/rmol-paths.hpp>
00025
00026 namespace boost_utf = boost::unit_test;
00027
00028 // (Boost) Unit Test XML Report
00029 std::ofstream utfReportStream ("OptimiseTestSuite_utfresults.xml");
00030
00034 struct UnitTestConfig {
00036     UnitTestConfig() {
00037         boost_utf::unit_test_log.set_stream (utfReportStream);
00038         boost_utf::unit_test_log.set_format (boost_utf::XML);
00039         boost_utf::unit_test_log.set_threshold_level (boost_utf::log_test_units);
00040         //boost_utf::unit_test_log.set_threshold_level
00041         (boost_utf::log_successful_tests);
00042     }
00043
00044     ~UnitTestConfig() {
00045     }
00046 };
00047
00048
00049 // ////////////////////////////////////////
00050 int testOptimiseHelper (const unsigned short optimisationMethodFlag,
00051                        const bool isBuiltin) {
00052
00053     // Return value
00054     int oExpectedBookingLimit = 0;
00055
00056     // Output log File
00057     std::ostringstream oStr;
00058     oStr << "OptimiseTestSuite_" << optimisationMethodFlag << ".log";
00059     const stdair::Filename_T lLogFilename (oStr.str());
00060
00061     // Number of random draws to be generated (best if greater than 100)
00062     const int K = 100000;
00063
00064     // Methods of optimisation (0 = Monte-Carlo, 1 = Dynamic Programming,
00065     // 2 = EMSR, 3 = EMSR-a, 4 = EMSR-b, 5 = EMSR-a with sellup prob.)
00066     const unsigned short METHOD_FLAG = optimisationMethodFlag;
00067

```



```

00068 // Cabin Capacity (it must be greater then 100 here)
00069 const double cabinCapacity = 100.0;
00070
00071 // Set the log parameters
00072 std::ofstream logOutputFile;
00073 // Open and clean the log outputfile
00074 logOutputFile.open (lLogFilename.c_str());
00075 logOutputFile.clear();
00076
00077 // Initialise the RMOL service
00078 const stdair::BasLogParams lLogParams (stdair::LOG::DEBUG, logOutputFile);
00079 RMOL::RMOL_Service rmolService (lLogParams);
00080
00081 // Check whether or not a (CSV) input file should be read
00082 if (isBuiltin == true) {
00083
00084     // Build the default sample BOM tree and build a dummy BOM tree.
00085     rmolService.buildSampleBom();
00086
00087 } else {
00088
00089     // Parse the optimisation data and build a dummy BOM tree
00090     const stdair::Filename_T lRMInputFileName (STDAIR_SAMPLE_DIR
00091 "rm02.csv");
00092     rmolService.parseAndLoad (cabinCapacity, lRMInputFileName);
00093
00094 switch (METHOD_FLAG) {
00095 case 0: {
00096     // DEBUG
00097     STDAIR_LOG_DEBUG ("Optimisation by Monte-Carlo (MC)");
00098
00099     // Calculate the optimal protections by the Monte Carlo
00100     // Integration approach
00101     rmolService.optimalOptimisationByMCIntegration (K);
00102     break;
00103 }
00104
00105 case 1: {
00106     // DEBUG
00107     STDAIR_LOG_DEBUG ("Optimisation by Dynamic Programming (DP)");
00108
00109     // Calculate the optimal protections by DP.
00110     rmolService.optimalOptimisationByDP ();
00111     break;
00112 }
00113
00114 case 2: {
00115     // DEBUG
00116     STDAIR_LOG_DEBUG ("Calculate the Bid-Price Vectors (BPV) by EMSR");
00117
00118     // Calculate the Bid-Price Vector by EMSR
00119     rmolService.heuristicOptimisationByEmsr ();
00120     break;
00121 }
00122
00123 case 3: {
00124     // DEBUG
00125     STDAIR_LOG_DEBUG ("Calculate the Authorisation Levels (AUs) by EMSRa");
00126
00127     // Calculate the protections by EMSR-a
00128     // Test the EMSR-a algorithm implementation
00129     rmolService.heuristicOptimisationByEmsrA ();
00130
00131     // Return a cumulated booking limit value to test
00132     // oExpectedBookingLimit = static_cast<int> (lBookingLimitVector.at(2));
00133     break;
00134 }
00135
00136 case 4: {
00137     // DEBUG
00138     STDAIR_LOG_DEBUG ("Calculate the Authorisation Levels (AUs) by EMSRb");
00139
00140     // Calculate the protections by EMSR-b
00141     rmolService.heuristicOptimisationByEmsrB ();
00142     break;
00143 }
00144
00145 default: rmolService.optimalOptimisationByMCIntegration (K);
00146 }
00147
00148 // Close the log file
00149 logOutputFile.close();
00150
00151 return oExpectedBookingLimit;
00152 }
00153

```

```

00154
00155 // //////////////// Main: Unit Test Suite ////////////////
00156
00157 // Set the UTF configuration (re-direct the output to a specific file)
00158 BOOST_GLOBAL_FIXTURE (UnitTestFixture);
00159
00160 // //////////////// Tests are based on the following input values ////////////////
00161 // Tests are based on the following input values
00162 // price; mean; standard deviation;
00163 // 1050; 17.3; 5.8;
00164 // 567; 45.1; 15.0;
00165 // 534; 39.6; 13.2;
00166 // 520; 34.0; 11.3;
00167 // ////////////////
00168
00173 BOOST_AUTO_TEST_SUITE (master_test_suite)
00174
00175
00178 BOOST_AUTO_TEST_CASE (rmol_optimisation_monte_carlo) {
00179
00180     // State whether the BOM tree should be built-in or parsed from an input file
00181     const bool isBuiltin = false;
00182
00183     BOOST_CHECK_NO_THROW (testOptimiseHelper(0, isBuiltin));
00184 }
00185
00189 BOOST_AUTO_TEST_CASE (rmol_optimisation_dynamic_programming) {
00190
00191     // State whether the BOM tree should be built-in or parsed from an input file
00192     const bool isBuiltin = false;
00193
00194     BOOST_CHECK_NO_THROW (testOptimiseHelper(1, isBuiltin));
00195 }
00196
00201 BOOST_AUTO_TEST_CASE (rmol_optimisation_emsr_bpv) {
00202
00203     // State whether the BOM tree should be built-in or parsed from an input file
00204     const bool isBuiltin = false;
00205
00206     BOOST_CHECK_NO_THROW (testOptimiseHelper(2, isBuiltin));
00207 }
00208
00213 BOOST_AUTO_TEST_CASE (rmol_optimisation_emsr_a) {
00214
00215     // State whether the BOM tree should be built-in or parsed from an input file
00216     const bool isBuiltin = false;
00217
00218     BOOST_CHECK_NO_THROW (testOptimiseHelper(3, isBuiltin));
00219     // const int lBookingLimit = testOptimiseHelper(3);
00220     // const int lExpectedBookingLimit = 61;
00221     // BOOST_CHECK_EQUAL (lBookingLimit, lExpectedBookingLimit);
00222     // BOOST_CHECK_MESSAGE (lBookingLimit == lExpectedBookingLimit,
00223     //     "The booking limit is " << lBookingLimit
00224     //     << ", but it is expected to be "
00225     //     << lExpectedBookingLimit);
00226 }
00227
00232 BOOST_AUTO_TEST_CASE (rmol_optimisation_emsr_b) {
00233
00234     // State whether the BOM tree should be built-in or parsed from an input file
00235     const bool isBuiltin = false;
00236
00237     BOOST_CHECK_NO_THROW (testOptimiseHelper(4, isBuiltin));
00238 }
00239
00243 BOOST_AUTO_TEST_CASE (rmol_optimisation_monte_carlo_built_in) {
00244
00245     // State whether the BOM tree should be built-in or parsed from an input file
00246     const bool isBuiltin = true;
00247
00248     BOOST_CHECK_NO_THROW (testOptimiseHelper(5, isBuiltin));
00249 }
00250
00254 BOOST_AUTO_TEST_CASE (rmol_optimisation_dynamic_programming_built_in) {
00255
00256     // State whether the BOM tree should be built-in or parsed from an input file
00257     const bool isBuiltin = true;
00258
00259     BOOST_CHECK_NO_THROW (testOptimiseHelper(6, isBuiltin));
00260 }
00261
00266 BOOST_AUTO_TEST_CASE (rmol_optimisation_emsr_bpv_built_in) {
00267
00268     // State whether the BOM tree should be built-in or parsed from an input file
00269     const bool isBuiltin = true;
00270
00271     BOOST_CHECK_NO_THROW (testOptimiseHelper(7, isBuiltin));

```

```

00272 }
00273
00278 BOOST_AUTO_TEST_CASE (rmol_optimisation_emsr_a_built_in) {
00279
00280     // State whether the BOM tree should be built-in or parsed from an input file
00281     const bool isBuiltin = true;
00282
00283     BOOST_CHECK_NO_THROW (testOptimiseHelper(8, isBuiltin));
00284 }
00285
00290 BOOST_AUTO_TEST_CASE (rmol_optimisation_emsr_b_built_in) {
00291
00292     // State whether the BOM tree should be built-in or parsed from an input file
00293     const bool isBuiltin = true;
00294
00295     BOOST_CHECK_NO_THROW (testOptimiseHelper(9, isBuiltin));
00296 }
00297
00298 // End the test suite
00299 BOOST_AUTO_TEST_SUITE_END ()
00300
00301

```

26.109 test/rmol/OptimiseTestSuite.hpp File Reference

```

#include <sstream>
#include <cppunit/extensions/HelperMacros.h>

```

Classes

- class [OptimiseTestSuite](#)

Functions

- [CPPUNIT_TEST_SUITE_REGISTRATION \(OptimiseTestSuite\)](#)

26.109.1 Function Documentation

26.109.1.1 CPPUNIT_TEST_SUITE_REGISTRATION (OptimiseTestSuite)

26.110 OptimiseTestSuite.hpp

```

00001 // STL
00002 #include <sstream>
00003 // CPPUNIT
00004 #include <cppunit/extensions/HelperMacros.h>
00005
00006 class OptimiseTestSuite : public CppUnit::TestFixture {
00007     CPPUNIT_TEST_SUITE (OptimiseTestSuite);
00008     CPPUNIT_TEST (testOptimiseMC);
00009     CPPUNIT_TEST (testOptimiseDP);
00010     CPPUNIT_TEST (testOptimiseEMSR);
00011     CPPUNIT_TEST (testOptimiseEMSRa);
00012     CPPUNIT_TEST (testOptimiseEMSRb);
00013     CPPUNIT_TEST (testOptimiseEMSRaWithSU);
00014     // CPPUNIT_TEST (errorCase);
00015     CPPUNIT_TEST_SUITE_END ();
00016 public:
00017
00019     void testOptimiseMC();
00020
00022     void testOptimiseDP();
00023
00026     void testOptimiseEMSR();
00027
00030     void testOptimiseEMSRa();
00031
00034     void testOptimiseEMSRb();
00035
00037     // void errorCase ();
00038

```

```

00040     OptimiseTestSuite ();
00041
00042 protected:
00043     std::stringstream _describeKey;
00044 };
00045
00046 CPPUNIT_TEST_SUITE_REGISTRATION (
    OptimiseTestSuite);

```

26.111 test/rmol/UnconstrainerTestSuite.cpp File Reference

26.112 UnconstrainerTestSuite.cpp

```

00001
00005 // //////////////////////////////////////
00006 // Import section
00007 // //////////////////////////////////////
00008 // STL
00009 #include <sstream>
00010 #include <fstream>
00011 #include <string>
00012 // Boost Unit Test Framework (UTF)
00013 #define BOOST_TEST_DYN_LINK
00014 #define BOOST_TEST_MAIN
00015 #define BOOST_TEST_MODULE UnconstrainerTestSuite
00016 #include <boost/test/unit_test.hpp>
00017 // StdAir
00018 #include <stdair/basic/BasLogParams.hpp>
00019 #include <stdair/basic/BasDBParams.hpp>
00020 #include <stdair/basic/BasFileMgr.hpp>
00021 #include <stdair/service/Logger.hpp>
00022 // RMOL
00023 #include <rmol/RMOL_Service.hpp>
00024
00025 namespace boost_utf = boost::unit_test;
00026
00027 // (Boost) Unit Test XML Report
00028 std::ofstream utfReportStream ("UnconstrainerTestSuite_utfresults.xml");
00029
00033 struct UnitTestConfig {
00035     UnitTestConfig() {
00036         boost_utf::unit_test_log.set_stream (utfReportStream);
00037         boost_utf::unit_test_log.set_format (boost_utf::XML);
00038         boost_utf::unit_test_log.set_threshold_level (boost_utf::log_test_units);
00039         //boost_utf::unit_test_log.set_threshold_level
        (boost_utf::log_successful_tests);
00040     }
00041
00043     ~UnitTestConfig() {
00044     }
00045 };
00046
00047
00048 // ////////////////////////////////// Main: Unit Test Suite //////////////////////////////////
00049
00050 // Set the UTF configuration (re-direct the output to a specific file)
00051 BOOST_GLOBAL_FIXTURE (UnitTestConfig);
00052
00057 BOOST_AUTO_TEST_SUITE (master_test_suite)
00058
00059
00062 BOOST_AUTO_TEST_CASE (rmol_unconstraining_em) {
00063     const bool lTestFlag = true; // testUnconstrainerHelper(0);
00064     BOOST_CHECK_EQUAL (lTestFlag, true);
00065     BOOST_CHECK_MESSAGE (lTestFlag == true,
00066         "The test has failed. Please see the log file for "
00067         "<< \"more details\"");
00068 }
00069
00070 // End the test suite
00071 BOOST_AUTO_TEST_SUITE_END()
00072
00073

```

26.113 test/rmol/UnconstrainerTestSuite.hpp File Reference

```

#include <sstream>
#include <cppunit/extensions/HelperMacros.h>

```

Classes

- class [UnconstrainerTestSuite](#)

Functions

- [CPPUNIT_TEST_SUITE_REGISTRATION](#) ([UnconstrainerTestSuite](#))

26.113.1 Function Documentation

26.113.1.1 CPPUNIT_TEST_SUITE_REGISTRATION (UnconstrainerTestSuite)

26.114 UnconstrainerTestSuite.hpp

```
00001 // STL
00002 #include <sstream>
00003 // CPPUNIT
00004 #include <cppunit/extensions/HelperMacros.h>
00005
00006 class UnconstrainerTestSuite : public
    CppUnit::TestFixture {
00007     CPPUNIT_TEST_SUITE (UnconstrainerTestSuite);
00008     CPPUNIT_TEST (testUnconstrainingByEM);
00009     CPPUNIT_TEST_SUITE_END ();
00010 public:
00011
00012     void testUnconstrainingByEM();
00013
00014     UnconstrainerTestSuite ();
00015
00016 protected:
00017     std::stringstream _describeKey;
00018 };
00019
00020 CPPUNIT_TEST_SUITE_REGISTRATION (
    UnconstrainerTestSuite);
```

Index

- ~DemandGeneratorList
 - RMOL::DemandGeneratorList, [58](#)
- ~FacRmolServiceContext
 - RMOL::FacRmolServiceContext, [63](#)
- ~HistoricalBooking
 - RMOL::HistoricalBooking, [68](#)
- ~HistoricalBookingHolder
 - RMOL::HistoricalBookingHolder, [70](#)
- ~RMOL_Service
 - RMOL::RMOL_Service, [80](#)
- _describeKey
 - ForecasterTestSuite, [65](#)
 - OptimiseTestSuite, [77](#)
 - UnconstrainerTestSuite, [87](#)
- addHistoricalBooking
 - RMOL::HistoricalBookingHolder, [72](#)
- BINDIR
 - rmol-paths.hpp, [158](#)
- BookingClassUnconstrainedDemandMap_T
 - RMOL, [54](#)
- BookingClassUnconstrainedDemandVectorMap_T
 - RMOL, [54](#)
- BookingVector_T
 - RMOL, [54](#)
- BucketHolderList_T
 - RMOL, [53](#)
- buildRemainingDCPList
 - RMOL::Utilities, [88](#)
- buildRemainingDCPList2
 - RMOL::Utilities, [88](#)
- buildSampleBom
 - RMOL::RMOL_Service, [81](#)
- buildVirtualClassListForLegBasedOptimisation
 - RMOL::Optimiser, [76](#)
- calculateExpectedDemand
 - RMOL::HistoricalBookingHolder, [72](#)
- cdfGaussianQ
 - RMOL::DPOptimiser, [59](#)
- CmdAbstract, [56](#)
- computeAggregatedVirtualClass
 - RMOL::EmsrUtils, [62](#)
- computeDistributionParameters
 - RMOL::Utilities, [88](#)
- computeEmsrValue
 - RMOL::EmsrUtils, [62](#)
- computeProtectionLevel
 - RMOL::EmsrUtils, [62](#)
- create
 - RMOL::FacRmolServiceContext, [63](#)
- createCumulativeFRAT5Curve
 - RMOL::DefaultMap, [57](#)
- csvDisplay
 - RMOL::RMOL_Service, [84](#)
- DATADIR
 - rmol-paths.hpp, [159](#)
- DATAROOTDIR
 - rmol-paths.hpp, [159](#)
- DEFAULT_DCP_LIST
 - RMOL, [55](#)
- DEFAULT_EPSILON
 - RMOL, [55](#)
- DEFAULT_PRECISION
 - RMOL, [55](#)
- DOCDIR
 - rmol-paths.hpp, [159](#)
- DemandGeneratorList
 - RMOL::DemandGeneratorList, [57](#)
- DemandGeneratorList_T
 - RMOL::DemandGeneratorList, [57](#)
- describe
 - RMOL::HistoricalBooking, [69](#)
 - RMOL::HistoricalBookingHolder, [72](#)
- display
 - RMOL::HistoricalBooking, [69](#)
 - RMOL::HistoricalBookingHolder, [72](#)
- DistributionParameterList_T
 - RMOL, [53](#)
- doc/local/authors.doc, [89](#)
- doc/local/codingrules.doc, [89](#)
- doc/local/copyright.doc, [89](#)
- doc/local/documentation.doc, [89](#)
- doc/local/features.doc, [89](#)
- doc/local/help_wanted.doc, [89](#)
- doc/local/howto_release.doc, [89](#)
- doc/local/index.doc, [89](#)
- doc/local/installation.doc, [89](#)
- doc/local/linking.doc, [89](#)
- doc/local/test.doc, [89](#)
- doc/local/users_guide.doc, [89](#)
- doc/local/verification.doc, [89](#)
- doc/tutorial/tutorial.doc, [89](#)
- EXEC_PREFIX
 - rmol-paths.hpp, [158](#)
- FRAT5Curve_T
 - RMOL, [54](#)
- FacRmolServiceContext
 - RMOL::FacRmolServiceContext, [63](#)
 - RMOL::RMOL_ServiceContext, [85](#)
- FacServiceAbstract, [63](#)
- FlagVector_T
 - RMOL, [54](#)
- ForecastException
 - RMOL::ForecastException, [66](#)
- forecastOnD
 - RMOL::RMOL_Service, [82](#)
- forecastUsingAdditivePickUp
 - RMOL::Forecaster, [64](#)

- forecastUsingMultiplicativePickUp
 - RMOL::Forecaster, 64
- ForecasterTestSuite, 64
 - _describeKey, 65
 - ForecasterTestSuite, 65
 - ForecasterTestSuite, 65
 - testQForecaster, 65
- generateDemandVector
 - RMOL::MCOptimiser, 74
- generateVariateList
 - RMOL::DemandGeneratorList, 58
- getCensorshipFlag
 - RMOL::HistoricalBookingHolder, 71
- getDemandMean
 - RMOL::HistoricalBookingHolder, 71
- getFlag
 - RMOL::HistoricalBooking, 68
- getHistoricalBooking
 - RMOL::HistoricalBookingHolder, 71
- getListOfToBeUnconstrainedFlags
 - RMOL::HistoricalBookingHolder, 71
- getNbOfBookings
 - RMOL::HistoricalBooking, 68
- getNbOfDepartedSimilarSegments
 - RMOL::Utilities, 89
- getNbOfFlights
 - RMOL::HistoricalBookingHolder, 70
- getNbOfSegmentAlreadyPassedThisDTD
 - RMOL::GuillotineBlockHelper, 66
- getNbOfUncensoredBookings
 - RMOL::HistoricalBookingHolder, 70
- getNbOfUncensoredData
 - RMOL::HistoricalBookingHolder, 70
- getStandardDeviation
 - RMOL::HistoricalBookingHolder, 71
- getUncensoredStandardDeviation
 - RMOL::HistoricalBookingHolder, 70
- getUnconstrainedDemand
 - RMOL::HistoricalBooking, 68
 - RMOL::HistoricalBookingHolder, 71
- getUnconstrainedDemandOnFirstElement
 - RMOL::HistoricalBookingHolder, 71
- getYieldFeatures
 - RMOL::RMOL_Service, 82
- HTMLDIR
 - rmol-paths.hpp, 159
- hasPassedThisDTD
 - RMOL::GuillotineBlockHelper, 66
- heuristicOptimisationByEmsr
 - RMOL::Emsr, 61
 - RMOL::Optimiser, 76
 - RMOL::RMOL_Service, 81
- heuristicOptimisationByEmsrA
 - RMOL::Emsr, 61
 - RMOL::Optimiser, 76
 - RMOL::RMOL_Service, 81
- heuristicOptimisationByEmsrB
 - RMOL::Emsr, 61
 - RMOL::Optimiser, 76
 - RMOL::RMOL_Service, 81
- HistoricalBooking
 - RMOL::HistoricalBooking, 67
- HistoricalBookingHolder
 - RMOL::HistoricalBookingHolder, 70
- HistoricalBookingVector_T
 - RMOL, 53
- INCLUDEDIR
 - rmol-paths.hpp, 159
- INFODIR
 - rmol-paths.hpp, 159
- init
 - RMOL::DefaultDCPList, 56
- instance
 - RMOL::FacRmolServiceContext, 63
- jsonExport
 - RMOL::RMOL_Service, 84
- LIBDIR
 - rmol-paths.hpp, 158
- LIBEXECDIR
 - rmol-paths.hpp, 158
- MANDIR
 - rmol-paths.hpp, 159
- main
 - rmol.cpp, 94
- operator<<
 - rmol.cpp, 93
- optimalOptimisationByDP
 - RMOL::DPOptimiser, 59
 - RMOL::Optimiser, 75
 - RMOL::RMOL_Service, 81
- optimalOptimisationByMCIntegration
 - RMOL::MCOptimiser, 74
 - RMOL::Optimiser, 75
 - RMOL::RMOL_Service, 81
- optimisationByMCIntegration
 - RMOL::MCOptimiser, 74
- OptimisationException
 - RMOL::OptimisationException, 75
- optimise
 - rmol.cpp, 94
 - RMOL::Optimiser, 76
 - RMOL::RMOL_Service, 81
- optimiseOnD
 - RMOL::RMOL_Service, 83
- optimiseOnDUsingAdvancedRMCooperation
 - RMOL::RMOL_Service, 83
- optimiseOnDUsingRMCooperation
 - RMOL::RMOL_Service, 83
- OptimiseTestSuite, 76
 - _describeKey, 77
 - OptimiseTestSuite, 77

- OptimiseTestSuite, [77](#)
- testOptimiseDP, [77](#)
- testOptimiseEMSR, [77](#)
- testOptimiseEMSRa, [77](#)
- testOptimiseEMSRb, [77](#)
- testOptimiseMC, [77](#)
- optimiseUsingOnDForecast
 - RMOL::Optimiser, [76](#)
- OverbookingException
 - RMOL::OverbookingException, [78](#)
- PACKAGE
 - rmol-paths.hpp, [158](#)
- PACKAGE_NAME
 - rmol-paths.hpp, [158](#)
- PACKAGE_VERSION
 - rmol-paths.hpp, [158](#)
- PDFDIR
 - rmol-paths.hpp, [159](#)
- PREFIXDIR
 - rmol-paths.hpp, [158](#)
- parseAndLoad
 - RMOL::RMOL_Service, [80](#)
- parseInputFileAndBuildBom
 - RMOL::InventoryParser, [73](#)
- projectAggregatedDemandOnLegCabins
 - RMOL::RMOL_Service, [83](#)
- projectOnDDemandOnLegCabinsUsingDA
 - RMOL::RMOL_Service, [83](#)
- projectOnDDemandOnLegCabinsUsingDYP
 - RMOL::RMOL_Service, [83](#)
- projectOnDDemandOnLegCabinsUsingYP
 - RMOL::RMOL_Service, [83](#)
- RMOL, [52](#)
 - BookingClassUnconstrainedDemandMap_T, [54](#)
 - BookingClassUnconstrainedDemandVectorMap_T, [54](#)
 - BookingVector_T, [54](#)
 - BucketHolderList_T, [53](#)
 - DEFAULT_DCP_LIST, [55](#)
 - DEFAULT_EPSILON, [55](#)
 - DEFAULT_PRECISION, [55](#)
 - DistributionParameterList_T, [53](#)
 - FRAT5Curve_T, [54](#)
 - FlagVector_T, [54](#)
 - HistoricalBookingVector_T, [53](#)
 - RMOL_ServicePtr_T, [54](#)
 - UnconstrainedDemandVector_T, [54](#)
- RMOL::DPOptimiser, [59](#)
 - cdfGaussianQ, [59](#)
 - optimalOptimisationByDP, [59](#)
- RMOL::DefaultDCPList, [56](#)
 - init, [56](#)
- RMOL::DefaultMap, [56](#)
 - createCumulativeFRAT5Curve, [57](#)
- RMOL::DemandGeneratorList, [57](#)
 - ~DemandGeneratorList, [58](#)
 - DemandGeneratorList, [57](#)
 - DemandGeneratorList_T, [57](#)
 - generateVariateList, [58](#)
- RMOL::Detruncator, [58](#)
 - retrieveUnconstrainedDemandForFirstDCP, [59](#)
 - unconstrainUsingAdditivePickUp, [58](#)
 - unconstrainUsingMultiplicativePickUp, [58, 59](#)
- RMOL::EMDetruncator, [60](#)
 - unconstrainUsingEMMethod, [60](#)
- RMOL::Emsr, [60](#)
 - heuristicOptimisationByEmsr, [61](#)
 - heuristicOptimisationByEmsrA, [61](#)
 - heuristicOptimisationByEmsrB, [61](#)
- RMOL::EmsrUtils, [61](#)
 - computeAggregatedVirtualClass, [62](#)
 - computeEmsrValue, [62](#)
 - computeProtectionLevel, [62](#)
- RMOL::FacRmolServiceContext, [62](#)
 - ~FacRmolServiceContext, [63](#)
 - create, [63](#)
 - FacRmolServiceContext, [63](#)
 - instance, [63](#)
- RMOL::ForecastException, [65](#)
 - ForecastException, [66](#)
- RMOL::Forecaster, [64](#)
 - forecastUsingAdditivePickUp, [64](#)
 - forecastUsingMultiplicativePickUp, [64](#)
- RMOL::GuillotineBlockHelper, [66](#)
 - getNbOfSegmentAlreadyPassedThisDTD, [66](#)
 - hasPassedThisDTD, [66](#)
- RMOL::HistoricalBooking, [67](#)
 - ~HistoricalBooking, [68](#)
 - describe, [69](#)
 - display, [69](#)
 - getFlag, [68](#)
 - getNbOfBookings, [68](#)
 - getUnconstrainedDemand, [68](#)
 - HistoricalBooking, [67](#)
 - setParameters, [68](#)
 - setUnconstrainedDemand, [68](#)
 - toStream, [68](#)
- RMOL::HistoricalBookingHolder, [69](#)
 - ~HistoricalBookingHolder, [70](#)
 - addHistoricalBooking, [72](#)
 - calculateExpectedDemand, [72](#)
 - describe, [72](#)
 - display, [72](#)
 - getCensorshipFlag, [71](#)
 - getDemandMean, [71](#)
 - getHistoricalBooking, [71](#)
 - getListOfToBeUnconstrainedFlags, [71](#)
 - getNbOfFlights, [70](#)
 - getNbOfUncensoredBookings, [70](#)
 - getNbOfUncensoredData, [70](#)
 - getStandardDeviation, [71](#)
 - getUncensoredStandardDeviation, [70](#)
 - getUnconstrainedDemand, [71](#)
 - getUnconstrainedDemandOnFirstElement, [71](#)
 - HistoricalBookingHolder, [70](#)

- setUnconstrainedDemand, 72
 - toStream, 72
- RMOL::InventoryParser, 73
 - parseInputFileAndBuildBom, 73
- RMOL::MCOptimiser, 73
 - generateDemandVector, 74
 - optimalOptimisationByMCIntegration, 74
 - optimisationByMCIntegration, 74
- RMOL::OptimisationException, 74
 - OptimisationException, 75
- RMOL::Optimiser, 75
 - buildVirtualClassListForLegBasedOptimisation, 76
 - heuristicOptimisationByEmsr, 76
 - heuristicOptimisationByEmsrA, 76
 - heuristicOptimisationByEmsrB, 76
 - optimalOptimisationByDP, 75
 - optimalOptimisationByMCIntegration, 75
 - optimise, 76
 - optimiseUsingOnDForecast, 76
- RMOL::OverbookingException, 78
 - OverbookingException, 78
- RMOL::RMOL_Service, 78
 - ~RMOL_Service, 80
 - buildSampleBom, 81
 - csvDisplay, 84
 - forecastOnD, 82
 - getYieldFeatures, 82
 - heuristicOptimisationByEmsr, 81
 - heuristicOptimisationByEmsrA, 81
 - heuristicOptimisationByEmsrB, 81
 - jsonExport, 84
 - optimalOptimisationByDP, 81
 - optimalOptimisationByMCIntegration, 81
 - optimise, 81
 - optimiseOnD, 83
 - optimiseOnDUsingAdvancedRMCooperation, 83
 - optimiseOnDUsingRMCooperation, 83
 - parseAndLoad, 80
 - projectAggregatedDemandOnLegCabins, 83
 - projectOnDDemandOnLegCabinsUsingDA, 83
 - projectOnDDemandOnLegCabinsUsingYP, 83
 - RMOL_Service, 79, 80
 - resetDemandInformation, 82, 83
 - setOnDForecast, 82
 - setUpStudyStatManager, 81
 - updateBidPrice, 84
- RMOL::RMOL_ServiceContext, 84
 - FacRmolServiceContext, 85
 - RMOL_Service, 85
- RMOL::UnconstrainingException, 87
 - UnconstrainingException, 88
- RMOL::Utilities, 88
 - buildRemainingDCPList, 88
 - buildRemainingDCPList2, 88
 - computeDistributionParameters, 88
 - getNbOfDepartedSimilarSegments, 89
- RMOL_Service
 - RMOL::RMOL_Service, 79, 80
 - RMOL::RMOL_ServiceContext, 85
 - RMOL_ServicePtr_T
 - RMOL, 54
 - readConfiguration
 - rmol.cpp, 93
 - resetDemandInformation
 - RMOL::RMOL_Service, 82, 83
 - retrieveUnconstrainedDemandForFirstDCP
 - RMOL::Detruncator, 59
 - rmol-paths.hpp
 - BINDIR, 158
 - DATADIR, 159
 - DATAROOTDIR, 159
 - DOCDIR, 159
 - EXEC_PREFIX, 158
 - HTMLDIR, 159
 - INCLUDEDIR, 159
 - INFODIR, 159
 - LIBDIR, 158
 - LIBEXECDIR, 158
 - MANDIR, 159
 - PACKAGE, 158
 - PACKAGE_NAME, 158
 - PACKAGE_VERSION, 158
 - PDFDIR, 159
 - PREFIXDIR, 158
 - SBINDIR, 159
 - STDAIR_SAMPLE_DIR, 159
 - SYSCONFDIR, 159
 - rmol.cpp
 - main, 94
 - operator<<, 93
 - optimise, 94
 - readConfiguration, 93
 - rmol/RMOL_Service.hpp, 161, 162
 - rmol/RMOL_Types.hpp, 164, 165
 - rmol/basic/BasConst.cpp, 89, 90
 - rmol/basic/BasConst_Curves.hpp, 91
 - rmol/basic/BasConst_General.hpp, 91, 92
 - rmol/basic/BasConst_RMOL_Service.hpp, 92
 - rmol/batches/rmol.cpp, 93, 95
 - rmol/bom/BucketHolderTypes.hpp, 98
 - rmol/bom/DPOptimiser.cpp, 99
 - rmol/bom/DPOptimiser.hpp, 102
 - rmol/bom/DistributionParameterList.hpp, 98, 99
 - rmol/bom/EMDetruncator.cpp, 103
 - rmol/bom/EMDetruncator.hpp, 104
 - rmol/bom/Emsr.cpp, 105
 - rmol/bom/Emsr.hpp, 107
 - rmol/bom/EmsrUtils.cpp, 108
 - rmol/bom/EmsrUtils.hpp, 109
 - rmol/bom/GuillotineBlockHelper.cpp, 110
 - rmol/bom/GuillotineBlockHelper.hpp, 111
 - rmol/bom/HistoricalBooking.cpp, 112
 - rmol/bom/HistoricalBooking.hpp, 113
 - rmol/bom/HistoricalBookingHolder.cpp, 114
 - rmol/bom/HistoricalBookingHolder.hpp, 118
 - rmol/bom/MCOptimiser.cpp, 119

- rmol/bom/MCOptimiser.hpp, 123
- rmol/bom/Utilities.cpp, 126
- rmol/bom/Utilities.hpp, 128
- rmol/bom/old/DemandGeneratorList.cpp, 124
- rmol/bom/old/DemandGeneratorList.hpp, 125
- rmol/command/Detruncator.cpp, 128, 129
- rmol/command/Detruncator.hpp, 136, 137
- rmol/command/Forecaster.cpp, 138
- rmol/command/Forecaster.hpp, 149
- rmol/command/InventoryParser.cpp, 151
- rmol/command/InventoryParser.hpp, 153
- rmol/command/Optimiser.cpp, 154
- rmol/command/Optimiser.hpp, 157
- rmol/config/rmol-paths.hpp, 158, 159
- rmol/factory/FacRmolServiceContext.cpp, 160
- rmol/factory/FacRmolServiceContext.hpp, 161
- rmol/service/RMOL_Service.cpp, 166, 167
- rmol/service/RMOL_ServiceContext.cpp, 190, 191
- rmol/service/RMOL_ServiceContext.hpp, 192
- RootException, 85
- SBINDIR
 - rmol-paths.hpp, 159
- STDAIR_SAMPLE_DIR
 - rmol-paths.hpp, 159
- SYSCONFDIR
 - rmol-paths.hpp, 159
- ServiceAbstract, 85
- setOnDForecast
 - RMOL::RMOL_Service, 82
- setParameters
 - RMOL::HistoricalBooking, 68
- setUnconstrainedDemand
 - RMOL::HistoricalBooking, 68
 - RMOL::HistoricalBookingHolder, 72
- setUpStudyStatManager
 - RMOL::RMOL_Service, 81
- stdair, 55
- StructAbstract, 86
- test/rmol/ForecasterTestSuite.cpp, 196
- test/rmol/ForecasterTestSuite.hpp, 197
- test/rmol/OptimiseTestSuite.cpp, 198
- test/rmol/OptimiseTestSuite.hpp, 201
- test/rmol/UnconstrainerTestSuite.cpp, 202
- test/rmol/UnconstrainerTestSuite.hpp, 202, 203
- test/rmol/bomsforforecaster.cpp, 193
- TestFixture, 86
- testOptimiseDP
 - OptimiseTestSuite, 77
- testOptimiseEMSR
 - OptimiseTestSuite, 77
- testOptimiseEMSRa
 - OptimiseTestSuite, 77
- testOptimiseEMSRb
 - OptimiseTestSuite, 77
- testOptimiseMC
 - OptimiseTestSuite, 77
- testQForecaster
 - ForecasterTestSuite, 65
- testUnconstrainingByEM
 - UnconstrainerTestSuite, 87
- toStream
 - RMOL::HistoricalBooking, 68
 - RMOL::HistoricalBookingHolder, 72
- unconstrainUsingAdditivePickUp
 - RMOL::Detruncator, 58
- unconstrainUsingEMMethod
 - RMOL::EMDetruncator, 60
- unconstrainUsingMultiplicativePickUp
 - RMOL::Detruncator, 58, 59
- UnconstrainedDemandVector_T
 - RMOL, 54
- UnconstrainerTestSuite, 86
 - _describeKey, 87
 - testUnconstrainingByEM, 87
 - UnconstrainerTestSuite, 87
 - UnconstrainerTestSuite, 87
- UnconstrainingException
 - RMOL::UnconstrainingException, 88
- updateBidPrice
 - RMOL::RMOL_Service, 84