

globus gssapi gsi  
10.2

Generated by Doxygen 1.7.6.1

Sat Feb 25 2012 05:20:03

## Contents

<b>1</b>	<b>Deprecated List</b>	<b>1</b>
<b>2</b>	<b>Module Index</b>	<b>1</b>
2.1	Modules . . . . .	1
<b>3</b>	<b>Module Documentation</b>	<b>2</b>
3.1	Functions for manipulating a buffer set . . . . .	2
3.2	GSI GSS-API Constants . . . . .	3
3.3	Activation . . . . .	4
3.3.1	Detailed Description . . . . .	4
3.3.2	Define Documentation . . . . .	4
3.4	GSS Req Flags . . . . .	5
3.4.1	Detailed Description . . . . .	6
3.4.2	Define Documentation . . . . .	6
3.4.3	Function Documentation . . . . .	7
3.5	GSS Ret Flags . . . . .	22
3.5.1	Detailed Description . . . . .	22
3.5.2	Define Documentation . . . . .	22

## 1 Deprecated List

### Global GSS\_C\_GLOBUS\_ACCEPT\_PROXY\_SIGNED\_BY\_LIMITED\_PROXY\_FLAG (p. 6)

We now accept proxies signed by limited proxies if they are limited or independent.

### Global GSS\_C\_GLOBUS\_LIMITED\_PROXY\_MANY\_FLAG (p. 7)

We now accept proxies signed by limited proxies if they are limited or independent.

## 2 Module Index

### 2.1 Modules

Here is a list of all modules:

<b>Functions for manipulating a buffer set</b>	<b>2</b>
<b>GSI GSS-API Constants</b>	<b>3</b>
<b>Activation</b>	<b>4</b>
<b>GSS Req Flags</b>	<b>5</b>
<b>GSS Ret Flags</b>	<b>22</b>

## **3 Module Documentation**

### **3.1 Functions for manipulating a buffer set**

## 3.2 GSI GSS-API Constants

## 3.3 Activation

### Defines

- `#define GLOBUS_GSI_GSSAPI_MODULE`

### 3.3.1 Detailed Description

Globus GSI GSSAPI uses standard Globus module activation and deactivation. Before any Globus GSI GSSAPI functions are called, the following function should be called:

```
globus_module_activate(GLOBUS_GSI_GSSAPI_MODULE)
```

This function returns `GLOBUS_SUCCESS` if Globus GSI GSSAPI was successfully initialized, and you are therefore allowed to subsequently call Globus GSI GSSAPI functions. Otherwise, an error code is returned, and Globus GSI GSSAPI functions should not subsequently be called. This function may be called multiple times.

To deactivate Globus GSI GSSAPI, the following function should be called:

```
globus_module_deactivate(GLOBUS_GSI_GSSAPI_MODULE)
```

This function should be called once for each time Globus GSI GSSAPI was activated.

Note that it is not mandatory to call the above functions.

### 3.3.2 Define Documentation

#### 3.3.2.1 `#define GLOBUS_GSI_GSSAPI_MODULE`

Module descriptor.

### 3.4 GSS Req Flags

Collaboration diagram for GSS Req Flags:



#### Modules

- **GSS Ret Flags**

#### Defines

- `#define GSS_C_GLOBUS_DONT_ACCEPT_LIMITED_PROXY_FLAG 8192`
- `#define GSS_C_GLOBUS_DELEGATE_LIMITED_PROXY_FLAG 4096`
- `#define GSS_C_GLOBUS_ACCEPT_PROXY_SIGNED_BY_LIMITED_PROXY_FLAG 32768`
- `#define GSS_C_GLOBUS_ALLOW_MISSING_SIGNING_POLICY 65536`
- `#define GSS_C_GLOBUS_FORCE_SSL3 131072`
- `#define GSS_C_GLOBUS_LIMITED_PROXY_MANY_FLAG 32768`

#### Functions

- `OM_uint32 gss_acquire_cred (OM_uint32 *, const gss_name_t, OM_uint32, const gss_OID_set, gss_cred_usage_t, gss_cred_id_t *, gss_OID_set *, OM_uint32 *)`
- `OM_uint32 gss_release_cred (OM_uint32 *, gss_cred_id_t *)`
- `OM_uint32 gss_accept_sec_context (OM_uint32 *, gss_ctx_id_t *, const gss_cred_id_t, const gss_buffer_t, const gss_channel_bindings_t, gss_name_t *, gss_OID *, gss_buffer_t, OM_uint32 *, OM_uint32 *, gss_cred_id_t *)`
- `OM_uint32 gss_delete_sec_context (OM_uint32 *, gss_ctx_id_t *, gss_buffer_t)`
- `OM_uint32 gss_context_time (OM_uint32 *, const gss_ctx_id_t, OM_uint32 *)`
- `OM_uint32 gss_get_mic (OM_uint32 *, const gss_ctx_id_t, gss_qop_t, const gss_buffer_t, gss_buffer_t)`
- `OM_uint32 gss_verify_mic (OM_uint32 *, const gss_ctx_id_t, const gss_buffer_t, const gss_buffer_t, gss_qop_t *)`
- `OM_uint32 gss_wrap (OM_uint32 *, const gss_ctx_id_t, int, gss_qop_t, const gss_buffer_t, int *, gss_buffer_t)`
- `OM_uint32 gss_unwrap (OM_uint32 *, const gss_ctx_id_t, const gss_buffer_t, gss_buffer_t, int *, gss_qop_t *)`
- `OM_uint32 gss_display_status (OM_uint32 *, OM_uint32, int, const gss_OID, OM_uint32 *, gss_buffer_t)`
- `OM_uint32 gss_indicate_mechs (OM_uint32 *, gss_OID_set *)`
- `OM_uint32 gss_compare_name (OM_uint32 *, const gss_name_t, const gss_name_t, int *)`
- `OM_uint32 gss_import_name (OM_uint32 *, const gss_buffer_t, const gss_OID, gss_name_t *)`
- `OM_uint32 gss_export_name (OM_uint32 *, const gss_name_t, gss_buffer_t)`
- `OM_uint32 gss_release_name (OM_uint32 *, gss_name_t *)`
- `OM_uint32 gss_release_buffer (OM_uint32 *, gss_buffer_t)`
- `OM_uint32 gss_release_oid_set (OM_uint32 *, gss_OID_set *)`

- OM\_uint32 **gss\_inquire\_cred** (OM\_uint32 \*, const gss\_cred\_id\_t, gss\_name\_t \*, OM\_uint32 \*, gss\_cred\_usage\_t \*, gss\_OID\_set \*)
- OM\_uint32 **gss\_inquire\_context** (OM\_uint32 \*, const gss\_ctx\_id\_t, gss\_name\_t \*, gss\_name\_t \*, OM\_uint32 \*, gss\_OID \*, OM\_uint32 \*, int \*, int \*)
- OM\_uint32 **gss\_wrap\_size\_limit** (OM\_uint32 \*, const gss\_ctx\_id\_t, int, gss\_qop\_t, OM\_uint32, OM\_uint32 \*)
- OM\_uint32 **gss\_export\_sec\_context** (OM\_uint32 \*, gss\_ctx\_id\_t \*, gss\_buffer\_t)
- OM\_uint32 **gss\_import\_sec\_context** (OM\_uint32 \*, const gss\_buffer\_t, gss\_ctx\_id\_t \*)
- OM\_uint32 **gss\_create\_empty\_oid\_set** (OM\_uint32 \*, gss\_OID\_set \*)
- OM\_uint32 **gss\_add\_oid\_set\_member** (OM\_uint32 \*, const gss\_OID, gss\_OID\_set \*)
- OM\_uint32 **gss\_test\_oid\_set\_member** (OM\_uint32 \*, const gss\_OID, const gss\_OID\_set, int \*)
- OM\_uint32 **gss\_duplicate\_name** (OM\_uint32 \*, const gss\_name\_t, gss\_name\_t \*)
- OM\_uint32 **gss\_sign** (OM\_uint32 \*, gss\_ctx\_id\_t, int, gss\_buffer\_t, gss\_buffer\_t)
- OM\_uint32 **gss\_verify** (OM\_uint32 \*, gss\_ctx\_id\_t, gss\_buffer\_t, gss\_buffer\_t, int \*)
- OM\_uint32 **gss\_unseal** (OM\_uint32 \*, gss\_ctx\_id\_t, gss\_buffer\_t, gss\_buffer\_t, int \*, int \*)
- OM\_uint32 **gss\_create\_empty\_buffer\_set** (OM\_uint32 \*, gss\_buffer\_set\_t \*)
- OM\_uint32 **gss\_add\_buffer\_set\_member** (OM\_uint32 \*, const gss\_buffer\_t, gss\_buffer\_set\_t \*)
- OM\_uint32 **gss\_release\_buffer\_set** (OM\_uint32 \*, gss\_buffer\_set\_t \*)
- OM\_uint32 **gss\_import\_cred** (OM\_uint32 \*, gss\_cred\_id\_t \*, const gss\_OID, OM\_uint32, const gss\_buffer\_t, OM\_uint32, OM\_uint32 \*)
- OM\_uint32 **gss\_export\_cred** (OM\_uint32 \*, const gss\_cred\_id\_t, const gss\_OID, OM\_uint32, gss\_buffer\_t)
- OM\_uint32 **gss\_init\_delegation** (OM\_uint32 \*, const gss\_ctx\_id\_t, const gss\_cred\_id\_t, const gss\_OID, const gss\_OID\_set, const gss\_buffer\_set\_t, const gss\_buffer\_t, OM\_uint32, OM\_uint32, gss\_buffer\_t)
- OM\_uint32 **gss\_accept\_delegation** (OM\_uint32 \*, const gss\_ctx\_id\_t, const gss\_OID\_set, const gss\_buffer\_set\_t, const gss\_buffer\_t, OM\_uint32, OM\_uint32, OM\_uint32 \*, gss\_cred\_id\_t \*, gss\_OID \*, gss\_buffer\_t)
- OM\_uint32 **gss\_inquire\_cred\_by\_oid** (OM\_uint32 \*, const gss\_cred\_id\_t, const gss\_OID, gss\_buffer\_set\_t \*)
- OM\_uint32 **gss\_set\_sec\_context\_option** (OM\_uint32 \*, gss\_ctx\_id\_t \*, const gss\_OID, const gss\_buffer\_t)

### 3.4.1 Detailed Description

These macros set the REQUESTED type of context - these should be set (or not) in the context's req\_flags (or in the context's ret\_flags if accept\_sec\_context is being called)

### 3.4.2 Define Documentation

#### 3.4.2.1 #define GSS\_C\_GLOBUS\_DONT\_ACCEPT\_LIMITED\_PROXY\_FLAG 8192

Set if you don't want a context to accept a limited proxy.

If this flag is set, and a limited proxy is received, the call will not be successful and the context will not be set up

#### 3.4.2.2 #define GSS\_C\_GLOBUS\_DELEGATE\_LIMITED\_PROXY\_FLAG 4096

Set if you want the delegated proxy to be a limited proxy.

#### 3.4.2.3 #define GSS\_C\_GLOBUS\_ACCEPT\_PROXY\_SIGNED\_BY\_LIMITED\_PROXY\_FLAG 32768

Set if you want to accept proxies signed by limited proxies.

**Deprecated** We now accept proxies signed by limited proxies if they are limited or independent.

#### 3.4.2.4 #define GSS\_C\_GLOBUS\_ALLOW\_MISSING\_SIGNING\_POLICY 65536

Set if you want to allow CA certs without a signing policy to verify.

#### 3.4.2.5 #define GSS\_C\_GLOBUS\_FORCE\_SSL3 131072

Set if you want to force SSLv3 instead of negotiating TLSv1 or SSLv3.

#### 3.4.2.6 #define GSS\_C\_GLOBUS\_LIMITED\_PROXY\_MANY\_FLAG 32768

**Deprecated** We now accept proxies signed by limited proxies if they are limited or independent.

### 3.4.3 Function Documentation

#### 3.4.3.1 OM\_uint32 gss\_acquire\_cred ( OM\_uint32 \* minor\_status, const gss\_name\_t desired\_name\_P, OM\_uint32 time\_req, const gss\_OID\_set desired\_mechs, gss\_cred\_usage\_t cred\_usage, gss\_cred\_id\_t \* output\_cred\_handle\_P, gss\_OID\_set \* actual\_mechs, OM\_uint32 \* time\_rec )

GSSAPI routine to acquire the local credential.

See the latest IETF draft/RFC on the GSS C bindings.

Gets the local credentials. The proxy\_init\_cred does most of the work of setting up the SSL\_ctx, getting the user's cert, key, etc.

The globusid will be obtained from the certificate. (Minus and /CN=proxy entries.)

#### Parameters

<i>minor_status</i>	Mechanism specific status code. In this implementation, the minor_status is a cast from a globus_result_t value, which is either GLOBUS_SUCCESS or a globus error object ID if an error occurred.
<i>desired_name_P</i>	Name of principle whose credentials should be acquired This parameter maps to the desired subject of the cert to be acquired as the credential. Possible values are: For a service cert: <service name>=""><fqdn> For a host cert: <fqdn> For a proxy cert: <subject name>=""> For a user cert: <subject name>=""> This parameter can be NULL, in which case the cert is chosen using a default search order of: host, proxy, user, service
<i>time_req</i>	Number of seconds that credentials should remain valid. This value can be GSS_C_INDEFINITE for an unlimited lifetime. NOTE: in the current implementation, this parameter is ignored, since you can't change the expiration of a signed cert.
<i>desired_mechs</i>	
<i>cred_usage</i>	
<i>output_cred_handle_P</i>	
<i>actual_mechs</i>	
<i>time_rec</i>	

#### 3.4.3.2 OM\_uint32 gss\_release\_cred ( OM\_uint32 \* minor\_status, gss\_cred\_id\_t \* cred\_handle\_P )

Release the GSS cred handle.

#### Parameters

<i>minor_status</i>	The minor status result - this is a globus_result_t cast to a OM_uint32. To access the globus error object use: globus_error_get((globus_result_t) *minor_status)
<i>cred_handle_P</i>	The gss cred handle to be released



## Returns

The major status - GSS\_S\_COMPLETE or GSS\_S\_FAILURE

3.4.3.3 OM\_uint32 gss\_accept\_sec\_context ( OM\_uint32 \* minor\_status, gss\_ctx\_id\_t \* context\_handle\_P, const gss\_cred\_id\_t acceptor\_cred\_handle, const gss\_buffer\_t input\_token, const gss\_channel\_bindings\_t input\_chan\_bindings, gss\_name\_t \* src\_name\_P, gss\_OID \* mech\_type, gss\_buffer\_t output\_token, OM\_uint32 \* ret\_flags, OM\_uint32 \* time\_rec, gss\_cred\_id\_t \* delegated\_cred\_handle\_P )

## Parameters

<i>minor_status</i>	
<i>context_handle_P</i>	
<i>acceptor_cred_handle</i>	
<i>input_token</i>	
<i>input_chan_bindings</i>	
<i>src_name_P</i>	
<i>mech_type</i>	
<i>output_token</i>	
<i>ret_flags</i>	Also used as req_flags for other functions
<i>time_rec</i>	
<i>delegated_cred_handle_P</i>	

## Returns

3.4.3.4 OM\_uint32 gss\_delete\_sec\_context ( OM\_uint32 \* minor\_status, gss\_ctx\_id\_t \* context\_handle\_P, gss\_buffer\_t output\_token )

Delete the GSS Security Context.

## Parameters

<i>minor_status</i>	The minor status result - this is a globus_result_t cast to a OM_uint32. The
<i>context_handle_P</i>	The context handle to be deleted
<i>output_token</i>	The

3.4.3.5 OM\_uint32 gss\_context\_time ( OM\_uint32 \* minor\_status, const gss\_ctx\_id\_t context\_handle, OM\_uint32 \* time\_rec )

## Parameters

<i>minor_status</i>	
<i>context_handle</i>	
<i>time_rec</i>	

## Returns

3.4.3.6 **OM\_uint32 gss\_get\_mic** ( OM\_uint32 \* *minor\_status*, const gss\_ctx\_id\_t *context\_handle*, gss\_qop\_t *qop\_req*, const gss\_buffer\_t *message\_buffer*, gss\_buffer\_t *message\_token* )

Calculates a cryptographic MIC (message integrity check) over an application message, and returns that MIC in the token.

The token and message can then be passed to the peer application which calls **gss\_verify\_mic** (p. 9) to verify the MIC.

#### Parameters

<i>minor_status</i>	
<i>context_handle</i>	
<i>qop_req</i>	
<i>message_buffer</i>	
<i>message_token</i>	

#### Returns

3.4.3.7 **OM\_uint32 gss\_verify\_mic** ( OM\_uint32 \* *minor\_status*, const gss\_ctx\_id\_t *context\_handle*, const gss\_buffer\_t *message\_buffer*, const gss\_buffer\_t *token\_buffer*, gss\_qop\_t \* *qop\_state* )

Check a MIC of the data.

#### Parameters

<i>minor_status</i>	
<i>context_handle</i>	
<i>message_buffer</i>	
<i>token_buffer</i>	
<i>qop_state</i>	

#### Returns

3.4.3.8 **OM\_uint32 gss\_wrap** ( OM\_uint32 \* *minor\_status*, const gss\_ctx\_id\_t *context\_handle*, int *conf\_req\_flag*, gss\_qop\_t *qop\_req*, const gss\_buffer\_t *input\_message\_buffer*, int \* *conf\_state*, gss\_buffer\_t *output\_message\_buffer* )

Wrap a message for integrity and protection.

We do this using the SSLv3 routines, by writing to the SSL bio, and pulling off the buffer from the back of the write BIO. But we can't do everything SSL might want, such as control messages, or segment the messages here, since we are forced to using the gssapi tokens, and can not communicate directly with our peer. So there maybe some failures which would work with true SSL.

#### Parameters

<i>minor_status</i>	
<i>context_handle</i>	
<i>conf_req_flag</i>	
<i>qop_req</i>	
<i>input_message_buffer</i>	
<i>conf_state</i>	

<i>output_ - message_buffer</i>	
-------------------------------------	--

#### Returns

**3.4.3.9 OM\_uint32 gss\_unwrap ( OM\_uint32 \* *minor\_status*, const gss\_ctx\_id\_t *context\_handle*, const gss\_buffer\_t *input\_message\_buffer*, gss\_buffer\_t *output\_message\_buffer*, int \* *conf\_state*, gss\_qop\_t \* *qop\_state* )**

GSSAPI routine to unwrap a buffer which may have been received and wrapped by wrap.c.

Return the data from the wrapped buffer. There may also be errors, such as integrity errors. Since we can not communicate directly with our peer, we can not do everything SSL could, i.e. return a token for example.

#### Parameters

<i>minor_status</i>	
<i>context_handle</i>	
<i>input_message_ - buffer</i>	
<i>output_ - message_buffer</i>	
<i>conf_state</i>	
<i>qop_state</i>	

#### Returns

**3.4.3.10 OM\_uint32 gss\_display\_status ( OM\_uint32 \* *minor\_status*, OM\_uint32 *status\_value*, int *status\_type*, const gss\_OID *mech\_type*, OM\_uint32 \* *message\_context*, gss\_buffer\_t *status\_string* )**

Calls the SSLeay error print routines to produce a printable message.

This may need some work, as the SSLeay error messages are more of a trace, and may not be the best for the user. Also don't take advantage of being called in a loop.

#### Parameters

<i>minor_status</i>	
<i>status_value</i>	
<i>status_type</i>	
<i>mech_type</i>	
<i>message_ - context</i>	
<i>status_string</i>	

#### Returns

**3.4.3.11 OM\_uint32 gss\_indicate\_mechs ( OM\_uint32 \* *minor\_status*, gss\_OID\_set \* *mech\_set* )**

Passes back the mech set of available mechs.

We only have one for now.

#### Parameters

<i>minor_status</i>	
<i>mech_set</i>	

#### Returns

**3.4.3.12** `OM_uint32 gss_compare_name ( OM_uint32 * minor_status, const gss_name_t name1_P, const gss_name_t name2_P, int * name_equal )`

Compare two names.

GSSAPI names in this implementation are pointers to x509 names.

#### Parameters

<i>minor_status</i>	currently is always set to GLOBUS_SUCCESS
<i>name1_P</i>	
<i>name2_P</i>	
<i>name_equal</i>	

#### Returns

currently always returns GSS\_S\_COMPLETE

**3.4.3.13** `OM_uint32 gss_import_name ( OM_uint32 * minor_status, const gss_buffer_t input_name_buffer, const gss_OID input_name_type, gss_name_t * output_name_P )`

Import a name into a gss\_name\_t

Creates a new gss\_name\_t which contains a mechanism-specific representation of the input name.

GSSAPI OpenSSL implements the following name types, based on the input\_name\_type OID:

- GSS\_C\_NT\_ANONYMOUS (input\_name\_buffer is ignored)
- GSS\_C\_NT\_HOSTBASED\_SERVICE (input\_name\_buffer contains a string "service@FQN" which will match /CN=service/FQDN)
- GSS\_C\_NT\_EXPORT\_NAME (input\_name\_buffer contains a string with the X509\_oneline representation of a name) like "/X=Y/Z=A...")
- GSS\_C\_NO\_OID or GSS\_C\_NT\_USER\_NAME (input\_name\_buffer contains an X.500 name formatted like "/X=Y/Z=A...")
- GLOBUS\_GSS\_C\_NT\_HOST\_IP (input\_name\_buffer contains a string "FQDN/ip-address" which will match names with the FQDN or the IP address)
- GLOBUS\_GSS\_C\_NT\_X509 (input buffer is an X509 struct from OpenSSL)

#### Parameters

<i>minor_status</i>	Minor status
<i>input_name_ - buffer</i>	Input name buffer which is interpreted based on the <i>input_name_type</i>
<i>input_name_ - type</i>	OID of the name
<i>output_name_P</i>	New gss_name_t value containing the name

#### Return values

<i>GSS_S_COMPLETE</i>	indicates that a valid name representation is output in output_name and described by the type value in output_name_type.
<i>GSS_S_BAD_NAMETYPE</i>	indicates that the input_name_type is unsupported by the applicable underlying GSS-API mechanism(s), so the import operation could not be completed.
<i>GSS_S_BAD_NAME</i>	indicates that the provided input_name_string is ill-formed in terms of the input_name_type, so the import operation could not be completed.
<i>GSS_S_BAD_MECH</i>	indicates that the input presented for import was an exported name object and that its enclosed mechanism type was not recognized or was unsupported by the GSS-API implementation.
<i>GSS_S_FAILURE</i>	indicates that the requested operation could not be performed for reasons unspecified at the GSS-API level.

**3.4.3.14** `OM_uint32 gss_export_name ( OM_uint32 * minor_status, const gss_name_t input_name_P, gss_buffer_t exported_name )`

Produces a mechanism-independent exported name object.

See section 3.2 of RFC 2743.

**3.4.3.15** `OM_uint32 gss_release_name ( OM_uint32 * minor_status, gss_name_t * name_P )`

Release the GSS Name.

#### Parameters

<i>minor_status</i>	The minor status result - this is a globus_result_t cast to a (OM_uint32 *).
<i>name_P</i>	The gss name to be released

#### Returns

The major status - GSS\_S\_COMPLETE or GSS\_S\_FAILURE

**3.4.3.16** `OM_uint32 gss_release_buffer ( OM_uint32 * minor_status, gss_buffer_t buffer )`

#### Parameters

<i>minor_status</i>	
<i>buffer</i>	

#### Returns

**3.4.3.17** `OM_uint32 gss_release_oid_set ( OM_uint32 * minor_status, gss_OID_set * mech_set )`

Release the OID set.

#### Parameters

<i>minor_status</i>	
<i>mech_set</i>	

#### Returns

**3.4.3.18** OM\_uint32 gss\_inquire\_cred ( OM\_uint32 \* *minor\_status*, const gss\_cred\_id\_t *cred\_handle\_P*, gss\_name\_t \* *name*, OM\_uint32 \* *lifetime*, gss\_cred\_usage\_t \* *cred\_usage*, gss\_OID\_set \* *mechanisms* )

Get information about the current credential.

We will also allow the return of the proxy file name, if the *minor\_status* is set to a value of 57056 0xdee0 This is done since there is no way to pass back the delegated credential file name.

When 57056 is seen, this will cause a new copy of this credential to be written, and it is the user's responsibility to free the file when done. The name will be a pointer to a char \* of the file name which must be freed. The *minor\_status* will be set to 57057 0xdee1 to indicate this.

DEE - this is a kludge, till the GSSAPI get a better way to return the name.

If the minor status is not changed from 57056 to 57057 assume it is not this gssapi, and a gss name was returned.

#### Parameters

<i>minor_status</i>	
<i>cred_handle_P</i>	
<i>name</i>	
<i>lifetime</i>	
<i>cred_usage</i>	
<i>mechanisms</i>	

#### Returns

**3.4.3.19** OM\_uint32 gss\_inquire\_context ( OM\_uint32 \* *minor\_status*, const gss\_ctx\_id\_t *context\_handle\_P*, gss\_name\_t \* *src\_name\_P*, gss\_name\_t \* *targ\_name\_P*, OM\_uint32 \* *lifetime\_rec*, gss\_OID \* *mech\_type*, OM\_uint32 \* *ctx\_flags*, int \* *locally\_initiated*, int \* *open* )

#### Parameters

<i>minor_status</i>	
<i>context_handle_P</i>	
<i>src_name_P</i>	
<i>targ_name_P</i>	
<i>lifetime_rec</i>	
<i>mech_type</i>	
<i>ctx_flags</i>	
<i>locally_initiated</i>	
<i>open</i>	

## Returns

**3.4.3.20** `OM_uint32 gss_wrap_size_limit ( OM_uint32 * minor_status, const gss_ctx_id_t context_handle, int conf_req_flag, gss_qop_t qop_req, OM_uint32 req_output_size, OM_uint32 * max_input_size )`

GSSAPI routine to take a buffer, calculate a MIC which is returned as a token.

We will use the SSL protocol here.

## Parameters

<i>minor_status</i>	
<i>context_handle</i>	
<i>conf_req_flag</i>	
<i>qop_req</i>	
<i>req_output_size</i>	
<i>max_input_size</i>	

## Returns

**3.4.3.21** `OM_uint32 gss_export_sec_context ( OM_uint32 * minor_status, gss_ctx_id_t * context_handle_P, gss_buffer_t interprocess_token )`

Saves the important info about the session, converts it to a token, then deletes the context.

## Parameters

<i>minor_status</i>	
<i>context_handle_P</i>	
<i>interprocess_token</i>	

## Returns

For SSL handle We need to save: version of this routine. cred\_usage, i.e. are we accept or initiate target/source or name Session: Protocol, cipher, and Master-Key Client-Random Server-Random tmp.key\_block: client and server Mac\_secrets write\_sequence read\_sequence write iv read iv

see SSL 3.0 draft <http://wp.netscape.com/eng/ssl3/index.html>

**3.4.3.22** `OM_uint32 gss_import_sec_context ( OM_uint32 * minor_status, const gss_buffer_t interprocess_token, gss_ctx_id_t * context_handle_P )`

GSSAPI routine to import the security context based on the input token.

See: <draft-ietf-cat-gssv2-cbind-04.txt>

**3.4.3.23** `OM_uint32 gss_create_empty_oid_set ( OM_uint32 * minor_status, gss_OID_set * oid_set )`

Creates an object identifier set containing no object identifiers, to which members may be subsequently added using the GSS\_Add\_OID\_set\_member() routine.

These routines are intended to be used to construct sets of mechanism object identifiers, for input to GSS\_Acquire\_cred().

#### Parameters

<i>minor_status</i>	
<i>oid_set</i>	

#### Returns

GSS\_S\_COMPLETE indicates successful completion GSS\_S\_FAILURE indicates that the operation failed

**3.4.3.24 OM\_uint32 gss\_add\_oid\_set\_member ( OM\_uint32 \* *minor\_status*, const gss\_OID *member\_oid*, gss\_OID\_set \* *oid\_set* )**

Adds an Object Identifier to an Object Identifier set.

This routine is intended for use in conjunction with GSS\_Create\_empty\_OID\_set() when constructing a set of mechanism OIDs for input to GSS\_Acquire\_cred().

#### Parameters

<i>minor_status</i>	
<i>member_oid</i>	
<i>oid_set</i>	

#### Returns

GSS\_S\_COMPLETE indicates successful completion GSS\_S\_FAILURE indicates that the operation failed

**3.4.3.25 OM\_uint32 gss\_test\_oid\_set\_member ( OM\_uint32 \* *minor\_status*, const gss\_OID *member*, const gss\_OID\_set *set*, int \* *present* )**

Interrogates an Object Identifier set to determine whether a specified Object Identifier is a member.

This routine is intended to be used with OID sets returned by GSS\_Indicate\_mechs(), GSS\_Acquire\_cred(), and GSS\_Inquire\_cred().

#### Parameters

<i>minor_status</i>	
<i>member</i>	
<i>set</i>	
<i>present</i>	

#### Returns

GSS\_S\_COMPLETE indicates successful completion GSS\_S\_FAILURE indicates that the operation failed

**3.4.3.26 OM\_uint32 gss\_duplicate\_name ( OM\_uint32 \* *minor\_status*, const gss\_name\_t *src\_name*, gss\_name\_t \* *dest\_name* )**

Copy a GSS name.



#### Parameters

<i>minor_status</i>	
<i>src_name</i>	
<i>dest_name</i>	

#### Returns

3.4.3.27 **OM\_uint32 gss\_sign** ( OM\_uint32 \* *minor\_status*, gss\_ctx\_id\_t *context\_handle*, int *qop\_req*, gss\_buffer\_t *message\_buffer*, gss\_buffer\_t *message\_token* )

Deprecated.

Does the same thing as `gss_get_mic` for V1 compatability.

#### Parameters

<i>minor_status</i>	
<i>context_handle</i>	
<i>qop_req</i>	
<i>message_buffer</i>	
<i>message_token</i>	

#### Returns

3.4.3.28 **OM\_uint32 gss\_verify** ( OM\_uint32 \* *minor\_status*, gss\_ctx\_id\_t *context\_handle*, gss\_buffer\_t *message\_buffer*, gss\_buffer\_t *token\_buffer*, int \* *qop\_state* )

Obsolete variant of `gss_verify` for V1 compatability Check a MIC of the data.

#### Parameters

<i>minor_status</i>	
<i>context_handle</i>	
<i>message_buffer</i>	
<i>token_buffer</i>	
<i>qop_state</i>	

#### Returns

3.4.3.29 **OM\_uint32 gss\_unseal** ( OM\_uint32 \* *minor\_status*, gss\_ctx\_id\_t *context\_handle*, gss\_buffer\_t *input\_message\_buffer*, gss\_buffer\_t *output\_message\_buffer*, int \* *conf\_state*, int \* *qop\_state* )

Obsolete variant of `gss_wrap` for V1 compatability allow for non 32 bit integer in `qop_state`.

Return the data from the wrapped buffer. There may also be errors, such as integrity errors. Since we can not communicate directly with our peer, we can not do everything SSL could, i.e. return a token for example.

#### Parameters

<i>minor_status</i>	
<i>context_handle</i>	
<i>input_message_</i> <i>_buffer</i>	
<i>output_</i> <i>message_buffer</i>	
<i>conf_state</i>	
<i>qop_state</i>	

#### Returns

**3.4.3.30** `OM_uint32 gss_create_empty_buffer_set ( OM_uint32 * minor_status, gss_buffer_set_t * buffer_set )`

Create a empty buffer set.

This function allocates and initializes a empty buffer set. The memory allocated in this function should be freed by a call to `gss_release_buffer_set`.

#### Parameters

<i>minor_status</i>	The minor status returned by this function. This paramter will be 0 upon success.
<i>buffer_set</i>	Pointer to a buffer set structure.

#### Returns

GSS\_S\_COMPLETE upon success GSS\_S\_FAILURE failure

#### See also

**`gss_add_buffer_set_member`** (p. 17)

**`gss_release_buffer_set`** (p. 18)

**3.4.3.31** `OM_uint32 gss_add_buffer_set_member ( OM_uint32 * minor_status, const gss_buffer_t member_buffer, gss_buffer_set_t * buffer_set )`

Add a buffer to a buffer set.

This function allocates a new `gss_buffer_t`, intializes it with the values in the `member_buffer` parameter.

#### Parameters

<i>minor_status</i>	The minor status returned by this function. This paramter will be 0 upon success.
<i>member_buffer</i>	Buffer to insert into the buffer set.
<i>buffer_set</i>	Pointer to a initialized buffer set structure.

#### Returns

GSS\_S\_COMPLETE upon success GSS\_S\_FAILURE failure

See also

**gss\_create\_empty\_buffer\_set** (p. 17)  
**gss\_release\_buffer\_set** (p. 18)

#### 3.4.3.32 OM\_uint32 gss\_release\_buffer\_set ( OM\_uint32 \* minor\_status, gss\_buffer\_set\_t \* buffer\_set )

Free all memory associated with a buffer set.

This function will free all memory associated with a buffer set. Note that it will also free all memory associated with the buffers in the buffer set.

##### Parameters

<i>minor_status</i>	The minor status returned by this function. This parameter will be 0 upon success.
<i>buffer_set</i>	Pointer to a buffer set structure. This pointer will point to a NULL value upon return.

##### Returns

GSS\_S\_COMPLETE upon success GSS\_S\_FAILURE failure

See also

**gss\_create\_empty\_buffer\_set** (p. 17)  
**gss\_add\_buffer\_set\_member** (p. 17)

#### 3.4.3.33 OM\_uint32 gss\_import\_cred ( OM\_uint32 \* minor\_status, gss\_cred\_id\_t \* output\_cred\_handle, const gss\_OID desired\_mech, OM\_uint32 option\_req, const gss\_buffer\_t import\_buffer, OM\_uint32 time\_req, OM\_uint32 \* time\_rec )

Import a credential that was exported by **gss\_export\_cred()** (p. 18).

This function will import credentials exported by **gss\_export\_cred()** (p. 18). It is intended to allow a multiple use application to checkpoint delegated credentials.

##### Parameters

<i>minor_status</i>	The minor status returned by this function. This parameter will be 0 upon success.
<i>output_cred_handle</i>	Upon success, this parameter will contain the imported credential. When no longer needed this credential should be freed using <b>gss_release_cred()</b> (p. 7).
<i>desired_mech</i>	This parameter may be used to specify the desired security mechanism. May be GSS_C_NO_OID.
<i>option_req</i>	This parameter indicates which option_req value was used to produce the import_buffer.
<i>import_buffer</i>	A buffer produced by <b>gss_export_cred()</b> .
<i>time_req</i>	The requested period of validity (seconds) for the imported credential. May be NULL.
<i>time_rec</i>	This parameter will contain the received period of validity of the imported credential upon success. May be NULL.

##### Returns

GSS\_S\_COMPLETE upon successful completion GSS\_S\_BAD\_MECH if the requested security mechanism is unavailable GSS\_S\_DEFECTIVE\_TOKEN if the import\_buffer is defective GSS\_S\_FAILURE upon general failure

#### 3.4.3.34 OM\_uint32 gss\_export\_cred ( OM\_uint32 \* minor\_status, const gss\_cred\_id\_t cred\_handle, const gss\_OID desired\_mech, OM\_uint32 option\_req, gss\_buffer\_t export\_buffer )

Saves the credential so it can be checkpointed and imported by **gss\_import\_cred**.

#### Parameters

<i>minor_status</i>	
<i>cred_handle</i>	
<i>desired_mech</i>	Should either be <code>gss_mech_globus_gssapi_openssl</code> or <code>NULL</code> (in which case <code>gss_mech_globus_gssapi_openssl</code> is assumed).
<i>option_req</i>	
<i>export_buffer</i>	

#### Returns

**3.4.3.35** `OM_uint32 gss_init_delegation ( OM_uint32 * minor_status, const gss_ctx_id_t context_handle, const gss_cred_id_t cred_handle, const gss_OID desired_mech, const gss_OID_set extension_oids, const gss_buffer_set_t extension_buffers, const gss_buffer_t input_token, OM_uint32 req_flags, OM_uint32 time_req, gss_buffer_t output_token )`

Initiate the delegation of a credential.

This functions drives the initiating side of the credential delegation process. It is expected to be called in tandem with the `gss_accept_delegation` function.

#### Parameters

<i>minor_status</i>	The minor status returned by this function. This paramter will be 0 upon success.
<i>context_handle</i>	The security context over which the credential is delegated.
<i>cred_handle</i>	The credential to be delegated. May be <code>GSS_C_NO_CREDENTIAL</code> in which case the credential associated with the security context is used.
<i>desired_mech</i>	The desired security mechanism. Currently not used. May be <code>GSS_C_NO_OID</code> .
<i>extension_oids</i>	A set of extension oids corresponding to buffers in the <code>extension_buffers</code> parameter below. The extensions specified will be added to the delegated credential. May be <code>GSS_C_NO_BUFFER_SET</code> .
<i>extension_buffers</i>	A set of extension buffers corresponding to oids in the <code>extension_oids</code> paramter above. May be <code>GSS_C_NO_BUFFER_SET</code> .
<i>input_token</i>	The token that was produced by a prior call to <code>gss_accept_delegation</code> . This parameter will be ignored the first time this function is called.
<i>req_flags</i>	Flags that modify the behavior of the function. Currently only <code>GSS_C_GLOBUS_SSL_COMPATIBLE</code> and <code>GSS_C_GLOBUS_LIMITED_DELEG_PROXY_FLAG</code> are checked for. The <code>GSS_C_GLOBUS_SSL_COMPATIBLE</code> flag results in tokens that aren't wrapped and <code>GSS_C_GLOBUS_LIMITED_DELEG_PROXY_FLAG</code> causes the delegated proxy to be limited (requires that no extensions are specified).
<i>time_req</i>	The requested period of validity (seconds) of the delegated credential. Passing a <code>time_req</code> of 0 cause the delegated credential to have the same lifetime as the credential that issued it.
<i>output_token</i>	A token that should be passed to <code>gss_accept_delegation</code> if the return value is <code>GSS_S_CONTINUE_NEEDED</code> .

## Returns

GSS\_S\_COMPLETE upon successful completion GSS\_S\_CONTINUE\_NEEDED if the function needs to be called again. GSS\_S\_FAILURE upon failure

**3.4.3.36** `OM_uint32 gss_accept_delegation ( OM_uint32 * minor_status, const gss_ctx_id_t context_handle, const gss_OID_set extension_oids, const gss_buffer_set_t extension_buffers, const gss_buffer_t input_token, OM_uint32 req_flags, OM_uint32 time_req, OM_uint32 * time_rec, gss_cred_id_t * delegated_cred_handle, gss_OID * mech_type, gss_buffer_t output_token )`

Accept a delegated credential.

This functions drives the accepting side of the credential delegation process. It is expected to be called in tandem with the `gss_init_delegation` function.

## Parameters

<i>minor_status</i>	The minor status returned by this function. This paramter will be 0 upon success.
<i>context_handle</i>	The security context over which the credential is delegated.
<i>extension_oids</i>	A set of extension oids corresponding to buffers in the <i>extension_buffers</i> paramter below. May be GSS_C_NO_BUFFER_SET. Currently not used.
<i>extension_buffers</i>	A set of extension buffers corresponding to oids in the <i>extension_oids</i> paramter above. May be GSS_C_NO_BUFFER_SET. Currently not used.
<i>input_token</i>	The token that was produced by a prior call to <code>gss_init_delegation</code> .
<i>req_flags</i>	Flags that modify the behavior of the function. Currently only GSS_C_GLOBUS_SSL_COMPATIBLE is checked for. This flag results in tokens that aren't wrapped.
<i>time_req</i>	The requested period of validity (seconds) of the delegated credential. Currently a noop.
<i>time_rec</i>	This parameter will contain the received period of validity of the delegated credential upon success. May be NULL.
<i>delegated_cred_handle</i>	This parameter will contain the delegated credential upon success.
<i>mech_type</i>	Returns the security mechanism upon success. Currently not implemented. May be NULL.
<i>output_token</i>	A token that should be passed to <code>gss_init_delegation</code> if the return value is GSS_S_CONTINUE_NEEDED.

## Returns

GSS\_S\_COMPLETE upon successful completion GSS\_S\_CONTINUE\_NEEDED if the function needs to be called again. GSS\_S\_FAILURE upon failure

**3.4.3.37** `OM_uint32 gss_inquire_cred_by_oid ( OM_uint32 * minor_status, const gss_cred_id_t cred_handle, const gss_OID desired_object, gss_buffer_set_t * data_set )`

NOTE: Checks both the cert in the credential and the certs in the cert chain for a valid extension that matches the desired OID.

The first one found is used, starting with the endpoint cert, and then searching the cert chain.

## Parameters

<i>minor_status</i>	
<i>cred_handle</i>	
<i>desired_object</i>	
<i>data_set</i>	

## Returns

**3.4.3.38** `OM_uint32 gss_set_sec_context_option ( OM_uint32 * minor_status, gss_ctx_id_t * context_handle, const gss_OID option, const gss_buffer_t value )`

GSSAPI routine to initiate the sending of a security context See: <draft-ietf-cat-gssv2-cbind-04.txt>

## Parameters

<i>minor_status</i>	
<i>context_handle</i>	
<i>option</i>	
<i>value</i>	

## Returns

### 3.5 GSS Ret Flags

Collaboration diagram for GSS Ret Flags:



#### Defines

- `#define GSS_C_GLOBUS_RECEIVED_LIMITED_PROXY_FLAG` 8192
- `#define GSS_C_GLOBUS_RECEIVED_LIMITED_PROXY_DURING_DELEGATION_FLAG` 4096

#### 3.5.1 Detailed Description

These macros set the RETURNED context type - these will be set (or not) in the context's `ret_flags`.

#### 3.5.2 Define Documentation

##### 3.5.2.1 `#define GSS_C_GLOBUS_RECEIVED_LIMITED_PROXY_FLAG` 8192

If the proxy received is a limited proxy, this flag will be set in the returned context flags (`ret_flags`)

##### 3.5.2.2 `#define GSS_C_GLOBUS_RECEIVED_LIMITED_PROXY_DURING_DELEGATION_FLAG` 4096

If the proxy received is a limited proxy received during delegation, this flag is set in the returned flags.