# globus gss assist Reference Manual

## 4.0

Generated by Doxygen 1.3.6

Tue Aug 11 14:13:36 2009

# Contents

# 1 Globus GSI GSS Assist

The GSS Assist code provides convenience functions for using the Globus GSS-API.

# 2 globus gss assist Module Index

## 2.1 globus gss assist Modules

Here is a list of all modules:

# 3 globus gss assist Module Documentation

## 3.1 Activation

Globus GSI GSS Assist uses standard Globus module activation and deactivation.

**Defines**

- #define GLOBUS_GSI_GSS_ASSIST_MODULE

### 3.1.1 Detailed Description

Globus GSI GSS Assist uses standard Globus module activation and deactivation.

Before any Globus GSS Assist functions are called, the following function must be called:

```
globus_module_activate(GLOBUS_GSI_GSS_ASSIST_MODULE);
```

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

To deactivate Globus GSS Assist, the following function must be called:

```
globus_module_deactivate(GLOBUS_GSI_GSS_ASSIST_MODULE)
```

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

### 3.1.2 Define Documentation

#### 3.1.2.1 #define GLOBUS_GSI_GSS_ASSIST_MODULE

Module descriptor.

## 3.2 Utility Functions

Utility functions for GSSAPI.

**Accept Security Context**

- OM_uint32 globus_gss_assist_accept_sec_context (OM_uint32 ∗minor_status, gss_ctx_id_t ∗context_-handle, const gss_cred_id_t cred_handle, char ∗∗src_name_char, OM_uint32 ∗ret_flags, int ∗user_to_-user_flag, int ∗token_status, gss_cred_id_t ∗delegated_cred_handle, int(∗gss_assist_get_token)(void ∗, void ∗∗, size_t ∗), void ∗gss_assist_get_context, int(∗gss_assist_send_token)(void ∗, void ∗, size_t), void ∗gss_-assist_send_context)

**Accept Security Context Asyncronous**

- OM_uint32 globus_gss_assist_accept_sec_context_async (OM_uint32 ∗minor_status, gss_ctx_id_t ∗context_handle, const gss_cred_id_t cred_handle, char ∗∗src_name_char, OM_uint32 ∗ret_flags, int ∗user_to_user_flag, void ∗input_buffer, size_t input_buffer_len, void ∗∗output_bufferp, size_t ∗output_-buffer_lenp, gss_cred_id_t ∗delegated_cred_handle)

**Acquire Credential**

- OM_uint32 globus_gss_assist_acquire_cred (OM_uint32 ∗minor_status, gss_cred_usage_t cred_usage, gss_cred_id_t ∗output_cred_handle)

**Acquire Credential Extension**

- OM_uint32 globus_gss_assist_acquire_cred_ext (OM_uint32 ∗minor_status, char ∗desired_name_char, OM_uint32 time_req, const gss_OID_set desired_mechs, gss_cred_usage_t cred_usage, gss_cred_id_-t ∗output_cred_handle, gss_OID_set ∗actual_mechs, OM_uint32 ∗time_rec)

**Display Status**

- OM_uint32 globus_gss_assist_display_status (FILE ∗fp, char ∗comment, OM_uint32 major_status, OM_-uint32 minor_status, int token_status)

**Display Status String**

- OM_uint32 globus_gss_assist_display_status_str (char ∗∗str, char ∗comment, OM_uint32 major_status, OM_uint32 minor_status, int token_status)

**Gridmap**

- int globus_gss_assist_gridmap (char ∗globusidp, char ∗∗useridp)

**User OK**

- int globus_gss_assist_userok (char ∗globusid, char ∗userid)

**Map Local User**

- int globus_gss_assist_map_local_user (char ∗local_user, char ∗∗globusidp)

**[NOHEADER]**

- OM_uint32 globus_gss_assist_import_sec_context (OM_uint32 ∗minor_status, gss_ctx_id_t ∗context_-handle, int ∗token_status, int fdp, FILE ∗fperr)

**Init Security Context**

- OM_uint32 globus_gss_assist_init_sec_context (OM_uint32 ∗minor_status, const gss_cred_id_t cred_-handle, gss_ctx_id_t ∗context_handle, char ∗target_name_char, OM_uint32 req_flags, OM_uint32 ∗ret_-flags, int ∗token_status, int(∗gss_assist_get_token)(void ∗, void ∗∗, size_t ∗), void ∗gss_assist_get_context, int(∗gss_assist_send_token)(void ∗, void ∗, size_t), void ∗gss_assist_send_context)

**Init Security Context Async**

- OM_uint32 globus_gss_assist_init_sec_context_async (OM_uint32 ∗minor_status, const gss_cred_id_-t cred_handle, gss_ctx_id_t ∗context_handle, char ∗target_name_char, OM_uint32 req_flags, OM_uint32 ∗ret_flags, void ∗input_buffer, size_t input_buffer_len, void ∗∗output_bufferp, size_t ∗output_buffer_lenp)

**Will Handle Restrictions**

- OM_uint32 globus_gss_assist_will_handle_restrictions (OM_uint32 ∗minor_status, gss_ctx_id_t ∗context_handle)

**Get Unwrap**

- OM_uint32 globus_gss_assist_get_unwrap (OM_uint32 ∗minor_status, const gss_ctx_id_t context_handle, char ∗∗data, size_t ∗length, int ∗token_status, int(∗gss_assist_get_token)(void ∗, void ∗∗, size_t ∗), void ∗gss_assist_get_context, FILE ∗fperr)

**Wrap**

- OM_uint32 globus_gss_assist_wrap_send (OM_uint32 ∗minor_status, const gss_ctx_id_t context_handle, char ∗data, size_t length, int ∗token_status, int(∗gss_assist_send_token)(void ∗, void ∗, size_t), void ∗gss_-assist_send_context, FILE ∗fperr)

**Defines**

- #define NI_MAXHOST 255

### 3.2.1 Detailed Description

Utility functions for GSSAPI.

### 3.2.2 Define Documentation

#### 3.2.2.1 #define NI_MAXHOST 255

Create a GSS Name structure from the given hostname. This function tries to resolve the given host name string to the canonical DNS name for the host.

**Parameters:**
>    *hostname*  The host name or numerical address to be resolved and transform into a GSS Name
>
>    *authorization_hostname*  The resulting GSS Name

**Returns:**
>    GLOBUS_SUCCESS on successful completion, a error object otherwise

### 3.2.3 Function Documentation

#### 3.2.3.1 OM_uint32 globus_gss_assist_accept_sec_context (OM_uint32 ∗ *minor_status*, gss_ctx_id_t ∗ *context_handle*, const gss_cred_id_t *cred_handle*, char ∗∗ *src_name_char*, OM_uint32 ∗ *ret_flags*, int ∗ *user_to_user_flag*, int ∗ *token_status*, gss_cred_id_t ∗ *delegated_cred_handle*, int(∗ *gss_assist_get_token*)(void ∗, void ∗∗, size_t ∗), void ∗ *gss_assist_get_context*, int(∗ *gss_assist_send_token*)(void ∗, void ∗, size_t), void ∗ *gss_assist_send_context*)

This routine accepts a GSSAPI security context and is called by the gram_gatekeeper. It isolates the GSSAPI from the rest of the gram code.

Initialize a gssapi security connection. Used by the server. The context_handle is returned, and there is one for each connection. This routine will take cake of the looping and token processing, using the supplied get_token and send_token routines.

**Parameters:**
>    *minor_status*  gssapi return code
>
>    *context_handle*  pointer to returned context.
>
>    *cred_handle*  the cred handle obtained by acquire_cred.
>
>    *src_name_char*  Pointer to char string repersentation of the client which contacted the server. Maybe NULL if not wanted. Should be freed when done.
>
>    *ret_flags*  Pointer to which services are available after the connection is established. Maybe NULL if not wanted. We will also use this to pass in flags to the globus version of gssapi_ssleay
>
>    *user_to_user_flag*  Pointer to flag to be set if the src_name is the same as our name. (Follwing are particular to this assist routine)
>
>    *token_status*  assist routine get/send token status
>
>    *delegated_cred_handle*  pointer to be set to the credential delegated by the client if delegation occurs during the security handshake
>
>    *gss_assist_get_token*  a get token routine
>
>    *gss_assist_get_context*  first arg for the get token routine
>
>    *gss_assist_send_token*  a send token routine
>
>    *gss_assist_send_context*  first arg for the send token routine

**Returns:**
>    GSS_S_COMPLETE on sucess Other gss errors on failure.

### 3.2.3.2   OM_uint32 globus_gss_assist_accept_sec_context_async (OM_uint32 ∗ *minor_status*, gss_ctx_id_-t ∗ *context_handle*, const gss_cred_id_t *cred_handle*, char ∗∗ *src_name_char*, OM_uint32 ∗ *ret_flags*, int ∗ *user_to_user_flag*, void ∗ *input_buffer*, size_t *input_buffer_len*, void ∗∗ *output_bufferp*, size_t ∗ *output_-buffer_lenp*, gss_cred_id_t ∗ *delegated_cred_handle*)

This is a asynchronous version of the globus_gss_assist_accept_sec_context() function. Instead of looping itself it passes in and out the read and written buffers and the calling application is responsible for doing the I/O directly.

**Parameters:**

   ***minor_status***  gssapi return code

   ***context_handle***  pointer to returned context.

   ***cred_handle***  the cred handle obtained by acquire_cred.

   ***src_name_char***  Pointer to char string repersentation of the client which contacted the server. Maybe NULL if not wanted. Should be freed when done.

   ***ret_flags***  Pointer to which services are available after the connection is established.  Maybe NULL if not wanted. We will also use this to pass in flags to the globus version of gssapi_ssleay

   ***user_to_user_flag***  Pointer to flag to be set if the src_name is the same as our name.

   ***input_buffer***  pointer to a buffer received from peer.

   ***input_buffer_len***  length of the buffer input_buffer.

   ***output_bufferp***  pointer to a pointer which will be filled in with a pointer to a allocated block of memory. If non-NULL the contents of this block should be written to the peer where they will be fed into the gss_assist_init_sec_context_async() function.

   ***output_buffer_lenp***  pointer to an integer which will be filled in with the length of the allocated output buffer pointed to by ∗output_bufferp.

   ***delegated_cred_handle***  pointer to be set to the credential delegated by the client if delegation occurs during the security handshake

**Returns:**

   GSS_S_COMPLETE on successful completion when this function does not need to be called again.

GSS_S_CONTINUE_NEEDED when ∗output_bufferp should be sent to the peer and a new input_buffer read and this function called again.

Other gss errors on failure.

### 3.2.3.3   OM_uint32 globus_gss_assist_acquire_cred (OM_uint32 ∗ *minor_status*, gss_cred_usage_t *cred_-usage*, gss_cred_id_t ∗ *output_cred_handle*)

Called once at the start of the process, to obtain the credentials the process is running under. The

**Parameters:**

   ***minor_status***  pointer for return code

   ***cred_usage***  GSS_C_INITIATE, GSS_C_ACCEPT, or GSS_C_BOTH

   ***output_cred_handle***  Pointer to the returned handle. This needs to be passed to many gss routines.

**Returns:**

   GSS_S_COMPLETE on sucess Other GSS return codes

### 3.2.3.4 OM_uint32 globus_gss_assist_acquire_cred_ext (OM_uint32 ∗ *minor_status*, char ∗ *desired_-name_char*, OM_uint32 *time_req*, const gss_OID_set *desired_mechs*, gss_cred_usage_t *cred_usage*, gss_-cred_id_t ∗ *output_cred_handle*, gss_OID_set ∗ *actual_mechs*, OM_uint32 ∗ *time_rec*)

Called once at the start of the process, to obtain the credentials the process is running under. All the parameters of the gss_acquire_cred, except the desired_name is a string of the form: [type:]name. This will be imported with the type.

**Returns:**
   GSS_S_COMPLETE on sucess Other GSS return codes

**See also:**
   globus_gsi_gss_acquire_cred

### 3.2.3.5 OM_uint32 globus_gss_assist_display_status (FILE ∗ *fp*, char ∗ *comment*, OM_uint32 *major_-status*, OM_uint32 *minor_status*, int *token_status*)

Display the messages for the major and minor status on the file pointed at by fp. Takes care of the overloaded major_status if there was a problem with the get_token or send_token routines.

**Parameters:**
   *fp*  a file pointer

   *comment*  String to print out before other error messages.

   *major_status*  The major status to display

   *minor_status*  The minor status to display

   *token_status*  token status to display

**Returns:**
   0

### 3.2.3.6 OM_uint32 globus_gss_assist_display_status_str (char ∗∗ *str*, char ∗ *comment*, OM_uint32 *major_-status*, OM_uint32 *minor_status*, int *token_status*)

Display the messages for the major and minor status and return a string with the messages. Takes care of the overloaded major_status if there was a problem with the get_token or send_token routines.

**Parameters:**
   *str*  pointer to char ∗ for returned string. Must be freed

   *comment*  String to print out before other error messages.

   *major_status*  The major status to display

   *minor_status*  The minor status to display

   *token_status*  token status to display

**Returns:**
   0

### 3.2.3.7 int globus_gss_assist_gridmap (char * *globusidp*, char ** *useridp*)

Routines callable from globus based code to map a globusID to a local unix user

GRIDMAP environment variable pointing at the map file. Defaults to ~/.gridmap

A gridmap file is required if being run as root. if being run as a user,it is not required, and defaults to the current user who is running the command.

This is the same file used by the gssapi_cleartext but will be used with other gssapi implementations which do not use the gridmap file.

**Parameters:**
    *globusidp* the GSSAPI name from the client who requested authentication

    *useridp* the resulting user ID name for the local system

**Returns:**
    0 on success -1 if bad arguments 1 on error

### 3.2.3.8 int globus_gss_assist_userok (char * *globusid*, char * *userid*)

Check to see if a particular globusid is authorized to access the given local user account.

**Parameters:**
    *globusid* the globus id in string form - this should be the user's subject

    *userid* the local account that access is sought for

**Returns:**
    0 on success (authorization allowed) -1 if bad arguments 1 on error

### 3.2.3.9 int globus_gss_assist_map_local_user (char * *local_user*, char ** *globusidp*)

Routine for returning the default globus ID associated with a local user name. This is somewhat of a hack since there is not a guarenteed one-to-one mapping. What we do is look for the first entry in the gridmap file that has the local user as the default login. If the user is not a default on any entry, we find the first entry in which the user exists as a secondary mapping.

**Parameters:**
    *local_user* the local username to find the DN for

    *globusidp* the first DN found that reverse maps from the local_user

**Returns:**
    0 on success, otherwise an error object identifier is returned. use globus_error_get to get the error object from the id. The resulting error object must be freed using globus_object_free when it is no longer needed.

**See also:**
    globus_error_get
    globus_object_free

### 3.2.3.10 OM_uint32 globus_gss_assist_import_sec_context (OM_uint32 * *minor_status*, gss_ctx_id_t * *context_handle*, int * *token_status*, int *fdp*, FILE * *fperr*)

Import the security context from a file

**Parameters:**

    *minor_status* GSSAPI return code. This is a Globus Error code (or GLOBUS_SUCCESS) cast to a OM_-uint32 pointer. If an erro has occurred, the resulting error (from calling globus_error_get on this variable) needs to be freed by the caller

    *context_handle* The imported context

    *token_status* Errors that occurred while reading from the file

    *fdp* the file descriptor pointing to a file containing the security context

    *fperr* FILE ∗ to write error messages

**Returns:**

    the major status

### 3.2.3.11 OM_uint32 globus_gss_assist_init_sec_context (OM_uint32 ∗ *minor_status*, const gss_cred_id_-t *cred_handle*, gss_ctx_id_t ∗ *context_handle*, char ∗ *target_name_char*, OM_uint32 *req_flags*, OM_uint32 ∗ *ret_flags*, int ∗ *token_status*, int(∗ *gss_assist_get_token*)(void ∗, void ∗∗, size_t ∗), void ∗ *gss_assist_get_-context*, int(∗ *gss_assist_send_token*)(void ∗, void ∗, size_t), void ∗ *gss_assist_send_context*)

Initialize a gssapi security connection. Used by the client. The context_handle is returned, and there is one for each connection. This routine will take cake of the looping and token processing, using the supplied get_token and send_token routines.

**Parameters:**

    *minor_status* GSSAPI return code. The new minor_status is a globus_result_t cast to an OM_uint32. If the call was successful, the minor status is equivalant to GLOBUS_SUCCESS. Otherwise, it is a globus error object ID that can be passed to globus_error_get to get the error object. The error object needs to be freed with globus_object_free.

    *cred_handle* the cred handle obtained by acquire_cred.

    *context_handle* pointer to returned context.

    *target_name_char* char string repersentation of the server to be contacted.

    *req_flags* request flags, such as GSS_C_DELEG_FLAG for delegation and the GSS_C_MUTUAL_FLAG for mutual authentication.

    *ret_flags* Pointer to which services are available after the connection is established. Maybe NULL if not wanted.

The Follwing are particular to this assist routine:

**Parameters:**

    *token_status* the assist routine's get/send token status

    *gss_assist_get_token* function pointer for getting the token

    *gss_assist_get_context* first argument passed to the gss_assist_get_token function

    *gss_assist_send_token* function pointer for setting the token

    *gss_assist_send_context* first argument passed to the gss_assist_set_token function pointer

**Returns:**

    The major status

### 3.2.3.12 OM_uint32 globus_gss_assist_init_sec_context_async (OM_uint32 ∗ *minor_status*, const gss_-cred_id_t *cred_handle*, gss_ctx_id_t ∗ *context_handle*, char ∗ *target_name_char*, OM_uint32 *req_flags*, OM_uint32 ∗ *ret_flags*, void ∗ *input_buffer*, size_t *input_buffer_len*, void ∗∗ *output_bufferp*, size_t ∗ *output_-buffer_lenp*)

This is a asynchronous version of the globus_gss_assist_init_sec_context() function. Instead of looping itself it passes in and out the read and written buffers and the calling application is responsible for doing the I/O directly.

**Parameters:**

    *minor_status* GSSAPI return code. The new minor status is a globus_result_t cast to a OM_uint32. If an error occurred (GSS_ERROR(major_status)) the minor_status is a globus error object id. The error object can be obtained via globus_error_get and should be destroyed with globus_object_free when no longer needed. If no error occurred, the minor status is equal to GLOBUS_SUCCESS.

    *cred_handle* the cred handle obtained by acquire_cred.

    *context_handle* pointer to returned context.

    *target_name_char* char string repersentation of the server to be contacted.

    *req_flags* request flags, such as GSS_C_DELEG_FLAG for delegation and the GSS_C_MUTUAL_FLAG for mutual authentication.

    *ret_flags* Pointer to which services are available after the connection is established. Maybe NULL if not wanted.

    *input_buffer* pointer to a buffer received from peer. Should be NULL on first call.

    *input_buffer_len* length of the buffer input_buffer. Should be zero on first call.

    *output_bufferp* pointer to a pointer which will be filled in with a pointer to a allocated block of memory. If non-NULL the contents of this block should be written to the peer where they will be fed into the gss_assist_init_sec_context_async() function.

    *output_buffer_lenp* pointer to an integer which will be filled in with the length of the allocated output buffer pointed to by ∗output_bufferp.

**Returns:**

    GSS_S_COMPLETE on successful completion when this function does not need to be called again.

GSS_S_CONTINUE_NEEDED when ∗output_bufferp should be sent to the peer and a new input_buffer read and this function called again.

Other gss errors on failure.

### 3.2.3.13 OM_uint32 globus_gss_assist_will_handle_restrictions (OM_uint32 ∗ *minor_status*, gss_ctx_id_t ∗ *context_handle*)

Sets the context to handle restrictions

**Parameters:**

    *minor_status* the resulting minor status from setting the context handle

    *context_handle* the context handle to set the minor status of

**Returns:**

    the major status from setting the context

### 3.2.3.14 OM_uint32 globus_gss_assist_get_unwrap (OM_uint32 ∗ *minor_status*, const gss_ctx_id_-t *context_handle*, char ∗∗ *data*, size_t ∗ *length*, int ∗ *token_status*, int(∗ *gss_assist_get_token*)(void ∗, void ∗∗, size_t ∗), void ∗ *gss_assist_get_context*, FILE ∗ *fperr*)

Gets a token using the specific tokenizing functions, and performs the GSS unwrap of that token

**See also:**
> gss_unwrap

**Parameters:**
> ***minor_status*** GSSAPI return code,

**See also:**
> gss_unwrap

**Parameters:**
> ***context_handle*** the context
>
> ***data*** pointer to be set to the unwrapped application data. This must be freed by the caller.
>
> ***length*** pointer to be set to the length of the *data* byte array.
>
> ***token_status*** assist routine get/send token status
>
> ***gss_assist_get_token*** a detokenizing routine
>
> ***gss_assist_get_context*** first arg for above routine
>
> ***fperr*** error stream to print to

**Returns:**
> GSS_S_COMPLETE on sucess Other gss errors on failure.

**3.2.3.15 OM_uint32 globus_gss_assist_wrap_send (OM_uint32 ∗ *minor_status*, const gss_ctx_id_-t *context_handle*, char ∗ *data*, size_t *length*, int ∗ *token_status*, int(∗ *gss_assist_send_token*)(void ∗, void ∗, size_t), void ∗ *gss_assist_send_context*, FILE ∗ *fperr*)**

**Parameters:**
> ***minor_status*** GSSAPI return code. If the call was successful, the minor status is equal to GLOBUS_-SUCCESS. Otherwise, it is an error object ID for which globus_error_get() and globus_object_free() can be used to get and destroy it.
>
> ***context_handle*** the context.
>
> ***data*** pointer to application data to wrap and send
>
> ***length*** length of the *data* array
>
> ***token_status*** assist routine get/send token status
>
> ***gss_assist_send_token*** a send_token routine
>
> ***gss_assist_send_context*** first arg for the send_token
>
> ***fperr*** file handle to write error message to.

**Returns:**
> GSS_S_COMPLETE on sucess Other gss errors on failure.

**See also:**
> gss_wrap()

## 3.3 GSI GSS Assist Constants

**Enumerations**

- enum globus_gsi_gss_assist_error_t {
  GLOBUS_GSI_GSS_ASSIST_ERROR_SUCCESS = 0,

GLOBUS_GSI_GSS_ASSIST_ERROR_WITH_ARGUMENTS = 1,

GLOBUS_GSI_GSS_ASSIST_ERROR_USER_ID_DOESNT_MATCH = 2,

GLOBUS_GSI_GSS_ASSIST_ERROR_IN_GRIDMAP_NO_USER_ENTRY = 3,

GLOBUS_GSI_GSS_ASSIST_ERROR_WITH_GRIDMAP = 4,

GLOBUS_GSI_GSS_ASSIST_ERROR_INVALID_GRIDMAP_FORMAT = 5,

GLOBUS_GSI_GSS_ASSIST_ERROR_ERRNO = 6,

GLOBUS_GSI_GSS_ASSIST_ERROR_WITH_INIT = 7,

GLOBUS_GSI_GSS_ASSIST_ERROR_WITH_WRAP = 8,

GLOBUS_GSI_GSS_ASSIST_ERROR_WITH_TOKEN = 9,

GLOBUS_GSI_GSS_ASSIST_ERROR_EXPORTING_CONTEXT = 10,

GLOBUS_GSI_GSS_ASSIST_ERROR_IMPORTING_CONTEXT = 11,

GLOBUS_GSI_GSS_ASSIST_ERROR_INITIALIZING_CALLOUT_HANDLE = 12,

GLOBUS_GSI_GSS_ASSIST_ERROR_WITH_CALLOUT_CONFIG = 13,

GLOBUS_GSI_GSS_ASSIST_CALLOUT_ERROR = 14,

GLOBUS_GSI_GSS_ASSIST_GSSAPI_ERROR = 15,

GLOBUS_GSI_GSS_ASSIST_GRIDMAP_LOOKUP_FAILED = 16,

GLOBUS_GSI_GSS_ASSIST_BUFFER_TOO_SMALL = 17,

GLOBUS_GSI_GSS_ASSIST_ERROR_CANONICALIZING_HOSTNAME = 18 }

### 3.3.1    Enumeration Type Documentation

#### 3.3.1.1    enum globus_gsi_gss_assist_error_t

GSI GSS Assist Error codes.

**Enumeration values:**

*GLOBUS_GSI_GSS_ASSIST_ERROR_SUCCESS*   Success.

*GLOBUS_GSI_GSS_ASSIST_ERROR_WITH_ARGUMENTS*   No user entry in gridmap file.

*GLOBUS_GSI_GSS_ASSIST_ERROR_USER_ID_DOESNT_MATCH*   Error user ID doesn't match.

*GLOBUS_GSI_GSS_ASSIST_ERROR_IN_GRIDMAP_NO_USER_ENTRY* Error with arguments passed to function.

*GLOBUS_GSI_GSS_ASSIST_ERROR_WITH_GRIDMAP*   Error querying gridmap file.

*GLOBUS_GSI_GSS_ASSIST_ERROR_INVALID_GRIDMAP_FORMAT*   Invalid gridmap file format.

*GLOBUS_GSI_GSS_ASSIST_ERROR_ERRNO*   System Error.

*GLOBUS_GSI_GSS_ASSIST_ERROR_WITH_INIT*   Error during context initialization.

*GLOBUS_GSI_GSS_ASSIST_ERROR_WITH_WRAP*   Error during message wrap.

*GLOBUS_GSI_GSS_ASSIST_ERROR_WITH_TOKEN*   Error with token.

*GLOBUS_GSI_GSS_ASSIST_ERROR_EXPORTING_CONTEXT*   Error exporting context.

*GLOBUS_GSI_GSS_ASSIST_ERROR_IMPORTING_CONTEXT*   Error importing context.

*GLOBUS_GSI_GSS_ASSIST_ERROR_INITIALIZING_CALLOUT_HANDLE* Error initializing callout handle.

*GLOBUS_GSI_GSS_ASSIST_ERROR_WITH_CALLOUT_CONFIG* Error reading callout configuration.

*GLOBUS_GSI_GSS_ASSIST_CALLOUT_ERROR*   Error invoking callout.

*GLOBUS_GSI_GSS_ASSIST_GSSAPI_ERROR*   A GSSAPI returned an error.

***GLOBUS_GSI_GSS_ASSIST_GRIDMAP_LOOKUP_FAILED*** Gridmap lookup failure.

***GLOBUS_GSI_GSS_ASSIST_BUFFER_TOO_SMALL*** Caller provided insufficient buffer space for local identity.

***GLOBUS_GSI_GSS_ASSIST_ERROR_CANONICALIZING_HOSTNAME*** Failed to obtain canonical host name.

## 3.4 Security Token Transport

Token routines using fread and fwrite.

### Token Get File Descriptor

- int globus_gss_assist_token_get_fd (void *arg, void **bufp, size_t *sizep)

### Token Send File Descriptor

- int globus_gss_assist_token_send_fd (void *arg, void *buf, size_t size)

### Token Send File Descriptor Without Length

- int globus_gss_assist_token_send_fd_without_length (void *arg, void *buf, size_t size)

### Token Send File Descriptor Flag EX

- int globus_gss_assist_token_send_fd_ex (void *exp, void *buf, size_t size)

### Token Get Nexus

- int globus_gss_assist_token_get_nexus (void *arg, void **bufp, size_t *sizep)

### Token Send Nexus

- int globus_gss_assist_token_send_nexus (void *arg, void *buf, size_t size)

### Token Send Nexus Without Length

- int globus_gss_assist_token_send_nexus_without_length (void *arg, void *buf, size_t size)

### Token Send Nexus EX

- int globus_gss_assist_token_send_nexus_ex (void *exp, void *buf, size_t size)

### 3.4.1 Detailed Description

Token routines using fread and fwrite.

Additional code has been added to detect tokens which are sent without a length field. These can currently be only SSL tokens. This does require some knowledge of the underlying GSSAPI, by the application, but is within the guidelines of the GSSAPI specifications.

The get routine will automaticly attempt this test, while a new send routine will check a flag. The old send routine will work as before, sending a 4-byte length.

### 3.4.2 Function Documentation

#### 3.4.2.1 int globus_gss_assist_token_get_fd (void ∗ *arg*, void ∗∗ *bufp*, size_t ∗ *sizep*)

Use a open file discriptor to get a token. This function provides parameter types that allow it to be passed to globus_gss_assist_init_sec_context and globus_gss_assist_accept_sec_context

**Parameters:**
    *arg* the FILE ∗ stream cast to a void pointer

    *bufp* the resulting token

    *sizep* the size (number of bytes) read into bufp

**Returns:**
    0 on success > 0 is internal return < 0 is the -errno

#### 3.4.2.2 int globus_gss_assist_token_send_fd (void ∗ *arg*, void ∗ *buf*, size_t *size*)

Write a token to the open file descriptor. WIll write it with a 4 byte length. This function provides parameter types that allow it to be passed to globus_gss_assist_init_sec_context and globus_gss_assist_accept_sec_context

**Parameters:**
    *arg* the FILE ∗ stream to send the token on

    *buf* the token

    *size* the size of the token in bytes

**Returns:**
    0 on success >0 on error <0 on errno error

#### 3.4.2.3 int globus_gss_assist_token_send_fd_without_length (void ∗ *arg*, void ∗ *buf*, size_t *size*)

Write a token to the open file descripter. Will write it without a length. so as to

#### 3.4.2.4 int globus_gss_assist_token_send_fd_ex (void ∗ *exp*, void ∗ *buf*, size_t *size*)

Write a token to the open file descripter. will look at the flag to determine if the length field need to be written.

**Parameters:**
    *exp* the globus_gss_assist_ex variable that holds the FILE ∗ stream and flags to bet set

    *buf* the token buffer to send

    *size* size of the token buffer

**Returns:**
    0 on success >0 on error <0 on errno error (-errno)

### 3.4.2.5  int globus_gss_assist_token_get_nexus (void ∗ *arg*, void ∗∗ *bufp*, size_t ∗ *sizep*)

Use a nexus socket to get the tokens.

Additional code has been added to detect tokens which are sent without a length field. These can currently be only SSL tokens. This does require some knowledge of the underlying GSSAPI, by the application, but is within the guidelines of the GSSAPI specifications.

The get routine will automaticly attempt this test, while a new send routine will check a flag. The old send routine will work as before, sending a 4-byte length.

**Parameters:**

> *arg*  the globus_io_handle_t to get the token from
>
> *bufp*  the buffer to read the token into
>
> *sizep*  the size of what gets read

**Returns:**

> 0 on success > 0 is internal return < 0 is the -errno returned from nexus

### 3.4.2.6  int globus_gss_assist_token_send_nexus (void ∗ *arg*, void ∗ *buf*, size_t *size*)

Write a token to the nexus io handle. This function provides parameter types that allow it to be passed to globus_-gss_assist_init_sec_context and globus_gss_assist_accept_sec_context

**Parameters:**

> *arg*  nexus io handle to send the token on
>
> *buf*  the token as a buffer
>
> *size*  the size of the buffer

**Returns:**

> 0 on success >0 on error <0 on errno error (-errno)

### 3.4.2.7  int globus_gss_assist_token_send_nexus_without_length (void ∗ *arg*, void ∗ *buf*, size_t *size*)

Send a token on a nexus IO handle. Using this function the length is not sent.

**See also:**

> globus_gss_assist_token_get_nexus() for further info.

### 3.4.2.8  int globus_gss_assist_token_send_nexus_ex (void ∗ *exp*, void ∗ *buf*, size_t *size*)

Write a token to the open file descriptor. will look at the flag to determine if the length field need to be written.

**Parameters:**

> *exp*  The globus_gss_assist_ex that the wraps the nexus IO handle to send the token on
>
> *buf*  the buffer holding the token
>
> *size*  the size of the buffer

**Returns:**

> 0 on success >0 on error <0 on errno error (-errno)

# Index