

# globus common Reference Manual

## 11.5

Generated by Doxygen 1.4.4

Wed Jul 21 13:45:31 2010

# Contents

<a href="#">1 globus common Module Index</a>	<a href="#">1</a>
<a href="#">2 globus common Directory Hierarchy</a>	<a href="#">2</a>
<a href="#">3 globus common Data Structure Index</a>	<a href="#">2</a>
<a href="#">4 globus common Module Documentation</a>	<a href="#">2</a>
<a href="#">5 globus common Directory Documentation</a>	<a href="#">25</a>
<a href="#">6 globus common Data Structure Documentation</a>	<a href="#">27</a>

## 1 globus common Module Index

### 1.1 globus common Modules

Here is a list of all modules:

<a href="#">Globus Callback</a>	<a href="#">2</a>
<a href="#">Globus Callback API</a>	<a href="#">2</a>
<a href="#">Globus Callback Spaces</a>	<a href="#">2</a>
<a href="#">Globus Callback Signal Handling</a>	<a href="#">6</a>
<a href="#">Globus Error API</a>	<a href="#">12</a>
<a href="#">Globus Errno Error API</a>	<a href="#">8</a>
<a href="#">Error Construction</a>	<a href="#">8</a>
<a href="#">Error Data Accessors and Modifiers</a>	<a href="#">10</a>
<a href="#">Error Handling Helpers</a>	<a href="#">11</a>
<a href="#">Globus Generic Error API</a>	<a href="#">13</a>
<a href="#">Error Construction</a>	<a href="#">13</a>
<a href="#">Error Data Accessors and Modifiers</a>	<a href="#">15</a>
<a href="#">Error Handling Helpers</a>	<a href="#">18</a>
<a href="#">Globus Thread API</a>	<a href="#">20</a>
<a href="#">URL String Parser</a>	<a href="#">21</a>

## 2 globus common Directory Hierarchy

### 2.1 globus common Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

[library](#) 25

## 3 globus common Data Structure Index

### 3.1 globus common Data Structures

Here are the data structures with brief descriptions:

[globus\\_url\\_t](#) (Parsed URLs ) 27

## 4 globus common Module Documentation

### 4.1 Globus Callback

#### 4.1.1 Detailed Description

### 4.2 Globus Callback API

### 4.3 Globus Callback Spaces

#### Defines

- `#define GLOBUS_CALLBACK_GLOBAL_SPACE`

#### Enumerations

- `enum globus_callback_space_behavior_t {  
    GLOBUS_CALLBACK_SPACE_BEHAVIOR_SINGLE,  
    GLOBUS_CALLBACK_SPACE_BEHAVIOR_SERIALIZED,  
    GLOBUS_CALLBACK_SPACE_BEHAVIOR_THREADED }`

#### Functions

- `globus_result_t globus_callback_space_init (globus_callback_space_t *space, globus_callback_space_attr_t attr)`
- `globus_result_t globus_callback_space_reference (globus_callback_space_t space)`
- `globus_result_t globus_callback_space_destroy (globus_callback_space_t space)`
- `globus_result_t globus_callback_space_attr_init (globus_callback_space_attr_t *attr)`
- `globus_result_t globus_callback_space_attr_destroy (globus_callback_space_attr_t attr)`
- `globus_result_t globus_callback_space_attr_set_behavior (globus_callback_space_attr_t attr, globus_callback_space_behavior_t behavior)`
- `globus_result_t globus_callback_space_attr_get_behavior (globus_callback_space_attr_t attr, globus_callback_space_behavior_t *behavior)`

- globus\_result\_t [globus\\_callback\\_space\\_get](#) (globus\_callback\_space\_t \*space)
- int [globus\\_callback\\_space\\_get\\_depth](#) (globus\_callback\_space\_t space)
- globus\_bool\_t [globus\\_callback\\_space\\_is\\_single](#) (globus\_callback\_space\_t space)

#### 4.3.1 Detailed Description

#### 4.3.2 Define Documentation

##### 4.3.2.1 #define GLOBUS\_CALLBACK\_GLOBAL\_SPACE

The 'global' space handle.

This is the default space handle implied if no spaces are explicitly created.

#### 4.3.3 Enumeration Type Documentation

##### 4.3.3.1 enum [globus\\_callback\\_space\\_behavior\\_t](#)

Callback space behaviors describe how a space behaves.

In a non-threaded build all spaces exhibit a behavior == `_BEHAVIOR_SINGLE`. Setting a specific behavior in this case is ignored.

In a threaded build, `_BEHAVIOR_SINGLE` retains all the rules and behaviors of a non-threaded build while `_BEHAVIOR_THREADED` makes the space act as the global space.

Setting a space's behavior to `_BEHAVIOR_SINGLE` guarantees that the poll protection will always be there and all callbacks are serialized and only kicked out when polled for. In a threaded build, it is still necessary to poll for callbacks in a `_BEHAVIOR_SINGLE` space. (`globus_cond_wait()` will take care of this for you also)

Setting a space's behavior to `_BEHAVIOR_SERIALIZED` guarantees that the poll protection will always be there and all callbacks are serialized. In a threaded build, it is NOT necessary to poll for callbacks in a `_BEHAVIOR_SERIALIZED` space. Callbacks in this space will be delivered as soon as possible, but only one outstanding (and unblocked) callback will be allowed at any time.

Setting a space's behavior to `_BEHAVIOR_THREADED` allows the user to have the poll protection provided by spaces when built non-threaded, yet, be fully threaded when built threaded (where poll protection is not needed)

##### Enumerator:

**`GLOBUS_CALLBACK_SPACE_BEHAVIOR_SINGLE`** The default behavior.

Indicates that you always want poll protection and single threaded behavior (callbacks need to be explicitly polled for)

**`GLOBUS_CALLBACK_SPACE_BEHAVIOR_SERIALIZED`** Indicates that you want poll protection and all callbacks to be serialized (but they do not need to be polled for in a threaded build).

**`GLOBUS_CALLBACK_SPACE_BEHAVIOR_THREADED`** Indicates that you only want poll protection.

#### 4.3.4 Function Documentation

##### 4.3.4.1 [globus\\_result\\_t globus\\_callback\\_space\\_init](#) ([globus\\_callback\\_space\\_t](#) \* *space*, [globus\\_callback\\_space\\_attr\\_t](#) *attr*)

Initialize a user space.

This creates a user space.

##### Parameters:

*space* storage for the initialized space handle. This must be destroyed with [globus\\_callback\\_space\\_destroy\(\)](#)

*attr* a space attr describing desired behaviors. If GLOBUS\_NULL, the default behavior of GLOBUS\_CALLBACK\_SPACE\_BEHAVIOR\_SINGLE is assumed. This attr is copied into the space, so it is acceptable to destroy the attr as soon as it is no longer needed

**Returns:**

- GLOBUS\_CALLBACK\_ERROR\_INVALID\_ARGUMENT on NULL space
- GLOBUS\_CALLBACK\_ERROR\_MEMORY\_ALLOC
- GLOBUS\_SUCCESS

**See also:**

[globus\\_condattr\\_setspace\(\)](#)

#### 4.3.4.2 **globus\_result\_t globus\_callback\_space\_reference (globus\_callback\_space\_t *space*)**

Take a reference to a space.

A library which has been 'given' a space to provide callbacks on would use this to take a reference on the user's space. This prevents mayhem should a user destroy a space before the library is done with it. This reference should be destroyed with [globus\\_callback\\_space\\_destroy\(\)](#) (think dup())

**Parameters:**

*space* space to reference

**Returns:**

- GLOBUS\_CALLBACK\_ERROR\_INVALID\_SPACE
- GLOBUS\_SUCCESS

#### 4.3.4.3 **globus\_result\_t globus\_callback\_space\_destroy (globus\_callback\_space\_t *space*)**

Destroy a reference to a user space.

This will destroy a reference to a previously initialized space. Space will not actually be destroyed until all callbacks registered with this space have been run and unregistered (if the user has a handle to that callback) AND all references (from [globus\\_callback\\_space\\_reference\(\)](#)) have been destroyed.

**Parameters:**

*space* space to destroy, previously initialized by [globus\\_callback\\_space\\_init\(\)](#) or referenced with [globus\\_callback\\_space\\_reference\(\)](#)

**Returns:**

- GLOBUS\_CALLBACK\_ERROR\_INVALID\_SPACE
- GLOBUS\_SUCCESS

**See also:**

[globus\\_callback\\_space\\_init\(\)](#)  
[globus\\_callback\\_space\\_reference\(\)](#)

#### 4.3.4.4 **globus\_result\_t globus\_callback\_space\_attr\_init (globus\_callback\_space\_attr\_t \* *attr*)**

Initialize a space attr.

Currently, the only attr to set is the behavior. The default behavior associated with this attr is GLOBUS\_CALLBACK\_SPACE\_BEHAVIOR\_SINGLE

**Parameters:**

*attr* storage for the initialized attr. Must be destroyed with [globus\\_callback\\_space\\_attr\\_destroy\(\)](#)

**Returns:**

- GLOBUS\_CALLBACK\_ERROR\_INVALID\_ARGUMENT on NULL *attr*
- GLOBUS\_CALLBACK\_ERROR\_MEMORY\_ALLOC
- GLOBUS\_SUCCESS

**4.3.4.5 globus\_result\_t globus\_callback\_space\_attr\_destroy (globus\_callback\_space\_attr\_t *attr*)**

Destroy a space attr.

**Parameters:**

*attr* attr to destroy, previously initialized with [globus\\_callback\\_space\\_attr\\_init\(\)](#)

**Returns:**

- GLOBUS\_CALLBACK\_ERROR\_INVALID\_ARGUMENT on NULL *attr*
- GLOBUS\_SUCCESS

**See also:**

[globus\\_callback\\_space\\_attr\\_init\(\)](#)

**4.3.4.6 globus\_result\_t globus\_callback\_space\_attr\_set\_behavior (globus\_callback\_space\_attr\_t *attr*, [globus\\_callback\\_space\\_behavior\\_t](#) *behavior*)**

Set the behavior of a space.

**Parameters:**

*attr* attr to associate behavior with  
*behavior* desired behavior

**Returns:**

- GLOBUS\_CALLBACK\_ERROR\_INVALID\_ARGUMENT
- GLOBUS\_SUCCESS

**See also:**

[globus\\_callback\\_space\\_behavior\\_t](#)

**4.3.4.7 globus\_result\_t globus\_callback\_space\_attr\_get\_behavior (globus\_callback\_space\_attr\_t *attr*, [globus\\_callback\\_space\\_behavior\\_t](#) \* *behavior*)**

Get the behavior associated with an attr.

Note: for a non-threaded build, this will always pass back a behavior == GLOBUS\_CALLBACK\_SPACE\_BEHAVIOR\_SINGLE.

**Parameters:**

*attr* attr on which to query behavior  
*behavior* storage for the behavior

**Returns:**

- GLOBUS\_CALLBACK\_ERROR\_INVALID\_ARGUMENT
- GLOBUS\_SUCCESS

#### 4.3.4.8 `globus_result_t globus_callback_space_get (globus_callback_space_t * space)`

Retrieve the space of a currently running callback.

##### Parameters:

*space* storage for the handle to the space currently running

##### Returns:

- GLOBUS\_CALLBACK\_ERROR\_INVALID\_ARGUMENT on NULL space
- GLOBUS\_CALLBACK\_ERROR\_NO\_ACTIVE\_CALLBACK
- GLOBUS\_SUCCESS

#### 4.3.4.9 `int globus_callback_space_get_depth (globus_callback_space_t space)`

Retrieve the current nesting level of a space.

##### Parameters:

*space* The space to query.

##### Returns:

- the current nesting level
- -1 on invalid space

#### 4.3.4.10 `globus_bool_t globus_callback_space_is_single (globus_callback_space_t space)`

See if the specified space is a single threaded behavior space.

##### Parameters:

*space* the space to query

##### Returns:

- GLOBUS\_TRUE if space's behavior is \_BEHAVIOR\_SINGLE
- GLOBUS\_FALSE otherwise

## 4.4 Globus Callback Signal Handling

### Defines

- #define [GLOBUS\\_SIGNAL\\_INTERRUPT](#)

### Functions

- `globus_result_t globus\_callback\_space\_register\_signal\_handler (int signum, globus_bool_t persist, globus_callback_func_t callback_func, void *callback_user_arg, globus_callback_space_t space)`
- `globus_result_t globus\_callback\_unregister\_signal\_handler (int signum, globus_callback_func_t unregister_callback, void *unreg_arg)`
- `void globus\_callback\_add\_wakeup\_handler (void(*wakeup)(void *), void *user_arg)`

#### 4.4.1 Detailed Description

#### 4.4.2 Define Documentation

##### 4.4.2.1 #define GLOBUS\_SIGNAL\_INTERRUPT

Use this to trap interrupts (SIGINT on unix).

In the future, this will also map to handle ctrl-C on win32.

#### 4.4.3 Function Documentation

##### 4.4.3.1 globus\_result\_t globus\_callback\_space\_register\_signal\_handler (int *signum*, globus\_bool\_t *persist*, globus\_callback\_func\_t *callback\_func*, void \* *callback\_user\_arg*, globus\_callback\_space\_t *space*)

Fire a callback when the specified signal is received.

Note that there is a tiny delay between the time this call returns and the signal is actually handled by this library. It is likely that, if the signal was received the instant the call returned, it will be lost (this is normally not an issue, since you would call this in your startup code anyway)

##### Parameters:

*signum* The signal to receive. The following signals are not allowed: SIGKILL, SIGSEGV, SIGABRT, SIGBUS, SIGFPE, SIGILL, SIGIOT, SIGPIPE, SIGEMT, SIGSYS, SIGTRAP, SIGSTOP, SIGCONT, and SIGWAITING

*persist* If GLOBUS\_TRUE, keep this callback registered for multiple signals. If GLOBUS\_FALSE, the signal handler will automatically be unregistered once the signal has been received.

*callback\_func* the user func to call when a signal is received

*callback\_user\_arg* user arg that will be passed to callback

*space* the space to deliver callbacks to.

##### Returns:

- GLOBUS\_CALLBACK\_ERROR\_INVALID\_SPACE
- GLOBUS\_CALLBACK\_ERROR\_INVALID\_ARGUMENT
- GLOBUS\_SUCCESS otherwise

##### 4.4.3.2 globus\_result\_t globus\_callback\_unregister\_signal\_handler (int *signum*, globus\_callback\_func\_t *unregister\_callback*, void \* *unreg\_arg*)

Unregister a signal handling callback.

##### Parameters:

*signum* The signal to unregister.

*unregister\_callback* the function to call when the callback has been canceled and there are no running instances of it (may be NULL). This will be delivered to the same space used in the register call.

*unreg\_arg* user arg that will be passed to callback

##### Returns:

- GLOBUS\_CALLBACK\_ERROR\_INVALID\_ARGUMENT if this signal was registered with *persist* == false, then there is a race between a signal actually being caught and therefor automatically unregistered and the attempt to manually unregister it. If that race occurs, you will receive this error just as you would for any signal not registered.
- GLOBUS\_SUCCESS otherwise

#### 4.4.3.3 void globus\_callback\_add\_wakeup\_handler (void(\*) (void \*) wakeup, void \* user\_arg)

Register a wakeup handler with callback library.

This is really only needed in non-threaded builds, but for cross builds should be used everywhere that a callback may sleep for an extended period of time.

An example use is for an io poller that sleeps indefinitely on select(). If the callback library receives a signal that it needs to deliver asap, it will call the wakeup handler(s). These wakeup handlers must run as though they were called from a signal handler (don't use any thread utilities). The io poll example will likely write a single byte to a pipe that select() is monitoring.

This handler will not be unregistered until the callback library is deactivated (via common).

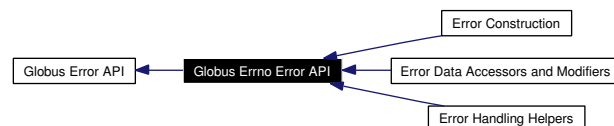
#### Parameters:

**wakeup** function to call when callback library needs you to return asap from any blocked callbacks.

**user\_arg** user data that will be passed along in the wakeup handler

## 4.5 Globus Errno Error API

Collaboration diagram for Globus Errno Error API:



These globus\_error functions are motivated by the desire to provide a easier way of generating new error types, while at the same time preserving all features (e.g.

#### Modules

- [Error Construction](#)
- [Error Data Accessors and Modifiers](#)
- [Error Handling Helpers](#)

#### 4.5.1 Detailed Description

These globus\_error functions are motivated by the desire to provide a easier way of generating new error types, while at the same time preserving all features (e.g.

memory management, chaining) of the current error handling framework. The functions in this API are auxiliary to the function in the Globus Generic Error API in the sense that they provide a wrapper for representing system errors in terms of a globus\_error\_t.

Any program that uses Globus Errno Error functions must include "globus\_common.h".

## 4.6 Error Construction

Collaboration diagram for Error Construction:



Create and initialize a Globus Errno Error object.

### Construct Error

- `globus_object_t * globus_error_construct_errno_error (globus_module_descriptor_t *base_source, globus_object_t *base_cause, const int system_errno)`

### Initialize Error

- `globus_object_t * globus_error_initialize_errno_error (globus_object_t *error, globus_module_descriptor_t *base_source, globus_object_t *base_cause, const int system_errno)`

### Defines

- `#define GLOBUS_ERROR_TYPE_ERRNO`

## 4.6.1 Detailed Description

Create and initialize a Globus Errno Error object.

This section defines operations to create and initialize Globus Errno Error objects.

## 4.6.2 Define Documentation

### 4.6.2.1 #define GLOBUS\_ERROR\_TYPE\_ERRNO

Error type definition.

## 4.6.3 Function Documentation

### 4.6.3.1 `globus_object_t* globus_error_construct_errno_error (globus_module_descriptor_t * base_source, globus_object_t * base_cause, const int system_errno)`

Allocate and initialize an error of type GLOBUS\_ERROR\_TYPE\_ERRNO.

#### Parameters:

*base\_source* Pointer to the originating module.

*base\_cause* The error object causing the error. If this is the original error, this parameter may be NULL.

*system\_errno* The system errno.

#### Returns:

The resulting error object. It is the user's responsibility to eventually free this object using `globus_object_free()`. A `globus_result_t` may be obtained by calling `globus_error_put()` on this object.

### 4.6.3.2 `globus_object_t* globus_error_initialize_errno_error (globus_object_t * error, globus_module_descriptor_t * base_source, globus_object_t * base_cause, const int system_errno)`

Initialize a previously allocated error of type GLOBUS\_ERROR\_TYPE\_ERRNO.

#### Parameters:

*error* The previously allocated error object.

*base\_source* Pointer to the originating module.

*base\_cause* The error object causing the error. If this is the original error this parameter may be NULL.

*system\_errno* The system errno.

**Returns:**

The resulting error object. You may have to call `globus_error_put()` on this object before passing it on.

## 4.7 Error Data Accessors and Modifiers

Collaboration diagram for Error Data Accessors and Modifiers:



Get and set data in a Globus Errno Error object.

### Get Errno

- int `globus_error_errno_get_errno` (globus\_object\_t \*error)

### Set Errno

- void `globus_error_errno_set_errno` (globus\_object\_t \*error, const int system\_errno)

#### 4.7.1 Detailed Description

Get and set data in a Globus Errno Error object.

This section defines operations for accessing and modifying data in a Globus Errno Error object.

#### 4.7.2 Function Documentation

##### 4.7.2.1 int `globus_error_errno_get_errno` (globus\_object\_t \* *error*)

Retrieve the system errno from a errno error object.

**Parameters:**

*error* The error from which to retrieve the errno

**Returns:**

The errno stored in the object

##### 4.7.2.2 void `globus_error_errno_set_errno` (globus\_object\_t \* *error*, const int *system\_errno*)

Set the errno in a errno error object.

**Parameters:**

*error* The error object for which to set the errno

*system\_errno* The system errno

**Returns:**

void

## 4.8 Error Handling Helpers

Collaboration diagram for Error Handling Helpers:



Helper functions for dealing with Globus Errno Error objects.

### Error Match

- `globus_bool_t globus_error_errno_match (globus_object_t *error, globus_module_descriptor_t *module, int system_errno)`

### Wrap Errno Error

- `globus_object_t * globus_error_wrap_errno_error (globus_module_descriptor_t *base_source, int system_errno, int type, const char *source_file, const char *source_func, int source_line, const char *short_desc_format,...)`

#### 4.8.1 Detailed Description

Helper functions for dealing with Globus Errno Error objects.

This section defines utility functions for dealing with Globus Errno Error objects.

#### 4.8.2 Function Documentation

##### 4.8.2.1 `globus_bool_t globus_error_errno_match (globus_object_t * error, globus_module_descriptor_t * module, int system_errno)`

Check whether the error originated from a specific module and matches a specific errno.

This function checks whether the error or any of its causative errors originated from a specific module and contains a specific errno. If the module descriptor is left unspecified this function will check for any error of the specified errno and vice versa.

##### Parameters:

*error* The error object for which to perform the check

*module* The module descriptor to check for

*system\_errno* The errno to check for

##### Returns:

GLOBUS\_TRUE - the error matched the module and errno GLOBUS\_FALSE - the error failed to match the module and errno

##### 4.8.2.2 `globus_object_t* globus_error_wrap_errno_error (globus_module_descriptor_t * base_source, int system_errno, int type, const char * source_file, const char * source_func, int source_line, const char * short_desc_format, ...)`

Allocate and initialize an error of type GLOBUS\_ERROR\_TYPE\_GLOBUS which contains a causal error of type GLOBUS\_ERROR\_TYPE\_ERRNO.

### Parameters:

***base\_source*** Pointer to the originating module.

***system\_errno*** The errno to use when generating the causal error.

**type** The error type. We may reserve part of this namespace for common errors. Errors not in this space are assumed to be local to the originating module.

*source\_file* Name of file. Use \_\_FILE\_\_

***source\_func*** Name of function. Use `_globus_func_name` and declare your func with `GlobusFuncName(<name>)`

*source\_line* Line number. Use `__LINE__`

***short\_desc\_format*** Short format string giving a succinct description of the error. To be passed on to the user.

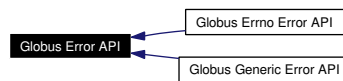
- ... Arguments for the format string.

**Returns:**

The resulting error object. It is the user's responsibility to eventually free this object using `globus_object_free()`. A `globus_result_t` may be obtained by calling `globus_error_put()` on this object.

## 4.9 Globus Error API

Collaboration diagram for Globus Error API:



Intended use:..

## Modules

- Globus Errno Error API
- Globus Generic Error API

### 4.9.1 Detailed Description

Intended use:.

If a function needs to return an error it should do the following:

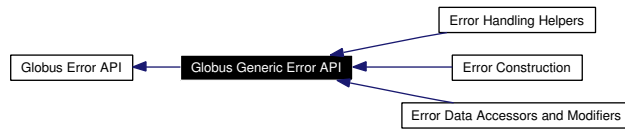
- External errors, such as error returns from system calls and GSSAPI errors, should be wrapped using the appropriate error type.
- The wrapped external error should then be passed as the cause of a globus error.
- External error types are expected to provide a utility function to combine the above two steps.
- The globus error should then be returned from the function.

### Notes on how to generate globus errors:

- Module specific error types should be greater or equal to 1024 (to leave some space for global error types).
- You may wish to generate a mapping from error types to format strings for use in short descriptions.
- You may also wish to generate a common prefix for all of the above format strings. The suggested prefix is "Function: s Line: s "

## 4.10 Globus Generic Error API

Collaboration diagram for Globus Generic Error API:



These `globus_error` functions are motivated by the desire to provide a easier way of generating new error types, while at the same time preserving all features (e.g.

### Modules

- [Error Construction](#)
- [Error Data Accessors and Modifiers](#)
- [Error Handling Helpers](#)

#### 4.10.1 Detailed Description

These `globus_error` functions are motivated by the desire to provide a easier way of generating new error types, while at the same time preserving all features (e.g.

memory management, chaining) of the current error handling framework. It does this by defining a generic error type for globus which in turn contains a integer in it's instance data which is used for carrying the actual error type information.

Any program that uses Globus Generic Error functions must include "globus\_common.h".

## 4.11 Error Construction

Collaboration diagram for Error Construction:



Create and initialize a Globus Generic Error object.

### Construct Error

- `globus_object_t * globus\_error\_construct\_error (globus_module_descriptor_t *base_source, globus_object_t *base_cause, int type, const char *source_file, const char *source_func, int source_line, const char *short_desc_format,...)`
- `globus_object_t * globus\_error\_v\_construct\_error (globus_module_descriptor_t *base_source, globus_object_t *base_cause, const int type, const char *source_file, const char *source_func, int source_line, const char *short_desc_format, va_list ap)`

### Initialize Error

- `globus_object_t * globus\_error\_initialize\_error (globus_object_t *error, globus_module_descriptor_t *base_source, globus_object_t *base_cause, int type, const char *source_file, const char *source_func, int source_line, const char *short_desc_format, va_list ap)`

## Defines

- #define [GLOBUS\\_ERROR\\_TYPE\\_GLOBUS](#)

### 4.11.1 Detailed Description

Create and initialize a Globus Generic Error object.

This section defines operations to create and initialize Globus Generic Error objects.

### 4.11.2 Define Documentation

#### 4.11.2.1 #define GLOBUS\_ERROR\_TYPE\_GLOBUS

Error type definition.

### 4.11.3 Function Documentation

**4.11.3.1 globus\_object\_t\* globus\_error\_construct\_error (globus\_module\_descriptor\_t \* *base\_source*, globus\_object\_t \* *base\_cause*, int *type*, const char \* *source\_file*, const char \* *source\_func*, int *source\_line*, const char \* *short\_desc\_format*, ...)**

Allocate and initialize an error of type GLOBUS\_ERROR\_TYPE\_GLOBUS.

#### Parameters:

*base\_source* Pointer to the originating module.

*base\_cause* The error object causing the error. If this is the original error this parameter may be NULL.

*type* The error type. We may reserve part of this namespace for common errors. Errors not in this space are assumed to be local to the originating module.

*source\_file* Name of file. Use `__FILE__`

*source\_func* Name of function. Use `_globus_func_name` and declare your func with `GlobusFuncName(<name>)`

*source\_line* Line number. Use `__LINE__`

*short\_desc\_format* Short format string giving a succinct description of the error. To be passed on to the user.

... Arguments for the format string.

#### Returns:

The resulting error object. It is the user's responsibility to eventually free this object using `globus_object_free()`. A `globus_result_t` may be obtained by calling `globus_error_put()` on this object.

**4.11.3.2 globus\_object\_t\* globus\_error\_v\_construct\_error (globus\_module\_descriptor\_t \* *base\_source*, globus\_object\_t \* *base\_cause*, const int *type*, const char \* *source\_file*, const char \* *source\_func*, int *source\_line*, const char \* *short\_desc\_format*, va\_list *ap*)**

Allocate and initialize an error of type GLOBUS\_ERROR\_TYPE\_GLOBUS.

#### Parameters:

*base\_source* Pointer to the originating module.

*base\_cause* The error object causing the error. If this is the original error this parameter may be NULL.

*type* The error type. We may reserve part of this namespace for common errors. Errors not in this space are assumed to be local to the originating module.

**source\_file** Name of file. Use `__FILE__`

**source\_func** Name of function. Use `_globus_func_name` and declare your func with `GlobusFuncName(<name>)`

**source\_line** Line number. Use `__LINE__`

**short\_desc\_format** Short format string giving a succinct description of the error. To be passed on to the user.

**ap** Arguments for the format string.

#### Returns:

The resulting error object. It is the user's responsibility to eventually free this object using `globus_object_free()`. A `globus_result_t` may be obtained by calling `globus_error_put()` on this object.

**4.11.3.3 globus\_object\_t\* globus\_error\_initialize\_error (globus\_object\_t \* error, globus\_module\_descriptor\_t \* base\_source, globus\_object\_t \* base\_cause, int type, const char \* source\_file, const char \* source\_func, int source\_line, const char \* short\_desc\_format, va\_list ap)**

Initialize a previously allocated error of type `GLOBUS_ERROR_TYPE_GLOBUS`.

#### Parameters:

**error** The previously allocated error object.

**base\_source** Pointer to the originating module.

**base\_cause** The error object causing the error. If this is the original error this parameter may be `NULL`.

**type** The error type. We may reserve part of this namespace for common errors. Errors not in this space are assumed to be local to the originating module.

**source\_file** Name of file. Use `__FILE__`

**source\_func** Name of function. Use `_globus_func_name` and declare your func with `GlobusFuncName(<name>)`

**source\_line** Line number. Use `__LINE__`

**short\_desc\_format** Short format string giving a succinct description of the error. To be passed on to the user.

**ap** Arguments for the format string.

#### Returns:

The resulting error object. You may have to call `globus_error_put()` on this object before passing it on.

## 4.12 Error Data Accessors and Modifiers

Collaboration diagram for Error Data Accessors and Modifiers:



Get and set data in a Globus Generic Error object.

#### Get Source

- `globus_module_descriptor_t * globus_error_get_source (globus_object_t *error)`

#### Set Source

- `void globus_error_set_source (globus_object_t *error, globus_module_descriptor_t *source_module)`

### Get Cause

- `globus_object_t * globus_error_get_cause (globus_object_t *error)`

### Set Cause

- `void globus_error_set_cause (globus_object_t *error, globus_object_t *causal_error)`

### Get Type

- `int globus_error_get_type (globus_object_t *error)`

### Set Type

- `void globus_error_set_type (globus_object_t *error, const int type)`

### Get Short Description

- `char * globus_error_get_short_desc (globus_object_t *error)`

### Set Short Description

- `void globus_error_set_short_desc (globus_object_t *error, const char *short_desc_format,...)`

### Get Long Description

- `char * globus_error_get_long_desc (globus_object_t *error)`

### Set Long Description

- `void globus_error_set_long_desc (globus_object_t *error, const char *long_desc_format,...)`

## 4.12.1 Detailed Description

Get and set data in a Globus Generic Error object.

This section defines operations for accessing and modifying data in a Globus Generic Error object.

## 4.12.2 Function Documentation

### 4.12.2.1 `globus_module_descriptor_t* globus_error_get_source (globus_object_t * error)`

Retrieve the originating module descriptor from a error object.

#### Parameters:

***error*** The error from which to retrieve the module descriptor

#### Returns:

The originating module descriptor.

#### **4.12.2.2 void globus\_error\_set\_source (globus\_object\_t \* *error*, globus\_module\_descriptor\_t \* *source\_module*)**

Set the originating module descriptor in a error object.

##### **Parameters:**

*error* The error object for which to set the causative error

*source\_module* The originating module descriptor

##### **Returns:**

void

#### **4.12.2.3 globus\_object\_t\* globus\_error\_get\_cause (globus\_object\_t \* *error*)**

Retrieve the underlying error from a error object.

##### **Parameters:**

*error* The error from which to retrieve the causative error.

##### **Returns:**

The underlying error object if it exists, NULL if it doesn't.

#### **4.12.2.4 void globus\_error\_set\_cause (globus\_object\_t \* *error*, globus\_object\_t \* *causal\_error*)**

Set the causative error in a error object.

##### **Parameters:**

*error* The error object for which to set the causative error.

*causal\_error* The causative error.

##### **Returns:**

void

#### **4.12.2.5 int globus\_error\_get\_type (globus\_object\_t \* *error*)**

Retrieve the error type from a generic globus error object.

##### **Parameters:**

*error* The error from which to retrieve the error type

##### **Returns:**

The error type of the object

#### **4.12.2.6 void globus\_error\_set\_type (globus\_object\_t \* *error*, const int *type*)**

Set the error type in a generic globus error object.

##### **Parameters:**

*error* The error object for which to set the error type

*type* The error type

##### **Returns:**

void

#### 4.12.2.7 `char* globus_error_get_short_desc (globus_object_t * error)`

Retrieve the short error description from a generic globus error object.

##### Parameters:

*error* The error from which to retrieve the description

##### Returns:

The short error description of the object

#### 4.12.2.8 `void globus_error_set_short_desc (globus_object_t * error, const char * short_desc_format, ...)`

Set the short error description in a generic globus error object.

##### Parameters:

*error* The error object for which to set the description

*short\_desc\_format* Short format string giving a succinct description of the error. To be passed on to the user.

... Arguments for the format string.

##### Returns:

void

#### 4.12.2.9 `char* globus_error_get_long_desc (globus_object_t * error)`

Retrieve the long error description from a generic globus error object.

##### Parameters:

*error* The error from which to retrieve the description

##### Returns:

The long error description of the object

#### 4.12.2.10 `void globus_error_set_long_desc (globus_object_t * error, const char * long_desc_format, ...)`

Set the long error description in a generic globus error object.

##### Parameters:

*error* The error object for which to set the description

*long\_desc\_format* Longer format string giving a more detailed explanation of the error.

##### Returns:

void

## 4.13 Error Handling Helpers

Collaboration diagram for Error Handling Helpers:



Helper functions for dealing with Globus Generic Error objects.

## Error Match

- `globus_bool_t globus_error_match (globus_object_t *error, globus_module_descriptor_t *module, int type)`

## Print Error Chain

- `char * globus_error_print_chain (globus_object_t *error)`

## Print User Friendly Error Message

- `char * globus_error_print_friendly (globus_object_t *error)`

### 4.13.1 Detailed Description

Helper functions for dealing with Globus Generic Error objects.

This section defines utility functions for dealing with Globus Generic Error objects.

### 4.13.2 Function Documentation

#### 4.13.2.1 `globus_bool_t globus_error_match (globus_object_t * error, globus_module_descriptor_t * module, int type)`

Check whether the error originated from a specific module and is of a specific type.

This function checks whether the error or any of its causative errors originated from a specific module and is of a specific type. If the module descriptor is left unspecified this function will check for any error of the specified type and vice versa.

#### Parameters:

*error* The error object for which to perform the check

*module* The module descriptor to check for

*type* The type to check for

#### Returns:

GLOBUS\_TRUE - the error matched the module and type GLOBUS\_FALSE - the error failed to match the module and type

#### 4.13.2.2 `char* globus_error_print_chain (globus_object_t * error)`

Return a string containing all printable errors found in a error object and its causative error chain.

If the GLOBUS\_ERROR\_VERBOSE env is set, file, line and function info will also be printed (where available). Otherwise, only the module name will be printed.

#### Parameters:

*error* The error to print

#### Returns:

A string containing all printable errors. This string needs to be freed by the user of this function.

#### 4.13.2.3 `char* globus_error_print_friendly (globus_object_t * error)`

Return a string containing error messages from the top 1 and bottom 3 objects, and, if found, show a friendly error message.

The error chain will be searched from top to bottom until a friendly handler is found and a friendly message is created.

If the `GLOBUS_ERROR_VERBOSE` env is set, then the result from [globus\\_error\\_print\\_chain\(\)](#) will be used.

##### Parameters:

*error* The error to print

##### Returns:

A string containing a friendly error message. This string needs to be freed by the user of this function.

## 4.14 Globus Thread API

### Functions

- int [globus\\_condattr\\_setspace](#) (globus\_condattr\_t \*attr, int space)
- int [globus\\_condattr\\_getspace](#) (globus\_condattr\_t \*attr, int \*space)

#### 4.14.1 Function Documentation

##### 4.14.1.1 `int globus_condattr_setspace (globus_condattr_t * attr, int space)`

Use this function to associate a space with a cond attr.

This will allow `globus_cond_wait` to poll the appropriate space (if applicable)

A condattr's default space is `GLOBUS_CALLBACK_GLOBAL_SPACE`

##### Parameters:

*attr* attr to associate space with.

*space* a previously initialized space

##### Returns:

- 0 on success

##### See also:

[Globus Callback Spaces](#)

##### 4.14.1.2 `int globus_condattr_getspace (globus_condattr_t * attr, int * space)`

Use this function to retrieve the space associated with a condattr.

##### Parameters:

*attr* attr to associate space with.

*space* storage for the space to be passed back

##### Returns:

- 0 on success

##### See also:

[Globus Callback Spaces](#)

## 4.15 URL String Parser

The Globus URL functions provide a simple mechanism for parsing a URL string into a data structure, and for determining the scheme of an URL string.

### Data Structures

- struct `globus_url_t`  
*Parsed URLs.*

### Enumerations

- enum `globus_url_scheme_t` {  
    `GLOBUS_URL_SCHEME_FTP` = 0,  
    `GLOBUS_URL_SCHEME_GSIFTP`,  
    `GLOBUS_URL_SCHEME_HTTP`,  
    `GLOBUS_URL_SCHEME_HTTPS`,  
    `GLOBUS_URL_SCHEME_LDAP`,  
    `GLOBUS_URL_SCHEME_FILE`,  
    `GLOBUS_URL_SCHEME_X_NEXUS`,  
    `GLOBUS_URL_SCHEME_X_GASS_CACHE`,  
    `GLOBUS_URL_SCHEME_UNKNOWN` ,  
    `GLOBUS_URL_NUM_SCHEMES` }

### Functions

- int `globus_url_parse` (const char \*url\_string, `globus_url_t` \*url)
- int `globus_url_parse_rfc1738` (const char \*url\_string, `globus_url_t` \*url)
- int `globus_url_parse_loose` (const char \*url\_string, `globus_url_t` \*url)
- int `globus_url_destroy` (`globus_url_t` \*url)
- int `globus_url_get_scheme` (const char \*url\_string, `globus_url_scheme_t` \*scheme\_type)
- int `globus_url_copy` (`globus_url_t` \*dst, const `globus_url_t` \*src)

#### 4.15.1 Detailed Description

The Globus URL functions provide a simple mechanism for parsing a URL string into a data structure, and for determining the scheme of an URL string.

These functions are part of the Globus common library. The `GLOBUS_COMMON` module must be activated in order to use them.

#### 4.15.2 Enumeration Type Documentation

##### 4.15.2.1 enum `globus_url_scheme_t`

URL Schemes.

The Globus URL library supports a set of URL schemes (protocols). This enumeration can be used to quickly dispatch a parsed URL based on a constant value.

See also:

[globus\\_url\\_t::scheme\\_type](#)

Enumerator:

***GLOBUS\_URL\_SCHEME\_FTP*** File Transfer Protocol.  
***GLOBUS\_URL\_SCHEME\_GSIFTP*** GSI-enhanced File Transfer Protocol.  
***GLOBUS\_URL\_SCHEME\_HTTP*** HyperText Transfer Protocol.  
***GLOBUS\_URL\_SCHEME\_HTTPS*** Secure HyperText Transfer Protocol.  
***GLOBUS\_URL\_SCHEME\_LDAP*** Lightweight Directory Access Protocol.  
***GLOBUS\_URL\_SCHEME\_FILE*** File Location.  
***GLOBUS\_URL\_SCHEME\_X\_NEXUS*** Nexus endpoint.  
***GLOBUS\_URL\_SCHEME\_X\_GASS\_CACHE*** GASS Cache Entry.  
***GLOBUS\_URL\_SCHEME\_UNKNOWN*** Any other URL of the form <scheme>://<something>.  
***GLOBUS\_URL\_NUM\_SCHEMES*** Total number of URL schemes supported.

### 4.15.3 Function Documentation

#### 4.15.3.1 `int globus_url_parse (const char * url_string, globus\_url\_t * url)`

Parse a string containing a URL into a [globus\\_url\\_t](#).

Parameters:

*url\_string* String to parse

*url* Pointer to [globus\\_url\\_t](#) to be filled with the fields of the url

Return values:

***GLOBUS\_SUCCESS*** The string was successfully parsed.  
***GLOBUS\_URL\_ERROR\_NULL\_STRING*** The *url\_string* was ***GLOBUS\_NULL***.  
***GLOBUS\_URL\_ERROR\_NULL\_URL*** The URL pointer was ***GLOBUS\_NULL***.  
***GLOBUS\_URL\_ERROR\_BAD\_SCHEME*** The URL scheme (protocol) contained invalid characters.  
***GLOBUS\_URL\_ERROR\_BAD\_USER*** The user part of the URL contained invalid characters.  
***GLOBUS\_URL\_ERROR\_BAD\_PASSWORD*** The password part of the URL contained invalid characters.  
***GLOBUS\_URL\_ERROR\_BAD\_HOST*** The host part of the URL contained invalid characters.  
***GLOBUS\_URL\_ERROR\_BAD\_PORT*** The port part of the URL contained invalid characters.  
***GLOBUS\_URL\_ERROR\_BAD\_PATH*** The path part of the URL contained invalid characters.  
***GLOBUS\_URL\_ERROR\_BAD\_DN*** -9 The DN part of an LDAP URL contained invalid characters.  
***GLOBUS\_URL\_ERROR\_BAD\_ATTRIBUTES*** -10 The attributes part of an LDAP URL contained invalid characters.  
***GLOBUS\_URL\_ERROR\_BAD\_SCOPE*** -11 The scope part of an LDAP URL contained invalid characters.  
***GLOBUS\_URL\_ERROR\_BAD\_FILTER*** -12 The filter part of an LDAP URL contained invalid characters.  
***GLOBUS\_URL\_ERROR\_OUT\_OF\_MEMORY*** -13 The library was unable to allocate memory to create the the [globus\\_url\\_t](#) contents.  
***GLOBUS\_URL\_ERROR\_INTERNAL\_ERROR*** -14 Some unexpected error occurred parsing the URL.

#### 4.15.3.2 `int globus_url_parse_rfc1738 (const char * url_string, globus\_url\_t * url)`

Parse a string containing a URL into a [globus\\_url\\_t](#).

##### Parameters:

*url\_string* String to parse

*url* Pointer to [globus\\_url\\_t](#) to be filled with the fields of the url

##### Return values:

**GLOBUS\_SUCCESS** The string was successfully parsed.

**GLOBUS\_URL\_ERROR\_NULL\_STRING** The *url\_string* was GLOBUS\_NULL.

**GLOBUS\_URL\_ERROR\_NULL\_URL** The URL pointer was GLOBUS\_NULL.

**GLOBUS\_URL\_ERROR\_BAD\_SCHEME** The URL scheme (protocol) contained invalid characters.

**GLOBUS\_URL\_ERROR\_BAD\_USER** The user part of the URL contained invalid characters.

**GLOBUS\_URL\_ERROR\_BAD\_PASSWORD** The password part of the URL contained invalid characters.

**GLOBUS\_URL\_ERROR\_BAD\_HOST** The host part of the URL contained invalid characters.

**GLOBUS\_URL\_ERROR\_BAD\_PORT** The port part of the URL contained invalid characters.

**GLOBUS\_URL\_ERROR\_BAD\_PATH** The path part of the URL contained invalid characters.

**GLOBUS\_URL\_ERROR\_BAD\_DN** -9 The DN part of an LDAP URL contained invalid characters.

**GLOBUS\_URL\_ERROR\_BAD\_ATTRIBUTES** -10 The attributes part of an LDAP URL contained invalid characters.

**GLOBUS\_URL\_ERROR\_BAD\_SCOPE** -11 The scope part of an LDAP URL contained invalid characters.

**GLOBUS\_URL\_ERROR\_BAD\_FILTER** -12 The filter part of an LDAP URL contained invalid characters.

**GLOBUS\_URL\_ERROR\_OUT\_OF\_MEMORY** -13 The library was unable to allocate memory to create the the [globus\\_url\\_t](#) contents.

**GLOBUS\_URL\_ERROR\_INTERNAL\_ERROR** -14 Some unexpected error occurred parsing the URL.

#### 4.15.3.3 `int globus_url_parse_loose (const char * url_string, globus\_url\_t * url)`

Parse a string containing a URL into a [globus\\_url\\_t](#) Looser restrictions on characters allowed in the path part of the URL.

##### Parameters:

*url\_string* String to parse

*url* Pointer to [globus\\_url\\_t](#) to be filled with the fields of the url

##### Return values:

**GLOBUS\_SUCCESS** The string was successfully parsed.

**GLOBUS\_URL\_ERROR\_NULL\_STRING** The *url\_string* was GLOBUS\_NULL.

**GLOBUS\_URL\_ERROR\_NULL\_URL** The URL pointer was GLOBUS\_NULL.

**GLOBUS\_URL\_ERROR\_BAD\_SCHEME** The URL scheme (protocol) contained invalid characters.

**GLOBUS\_URL\_ERROR\_BAD\_USER** The user part of the URL contained invalid characters.

**GLOBUS\_URL\_ERROR\_BAD\_PASSWORD** The password part of the URL contained invalid characters.

**GLOBUS\_URL\_ERROR\_BAD\_HOST** The host part of the URL contained invalid characters.

**GLOBUS\_URL\_ERROR\_BAD\_PORT** The port part of the URL contained invalid characters.

**GLOBUS\_URL\_ERROR\_BAD\_PATH** The path part of the URL contained invalid characters.

**GLOBUS\_URL\_ERROR\_BAD\_DN** -9 The DN part of an LDAP URL contained invalid characters.

***GLOBUS\_URL\_ERROR\_BAD\_ATTRIBUTES*** -10 The attributes part of an LDAP URL contained invalid characters.

***GLOBUS\_URL\_ERROR\_BAD\_SCOPE*** -11 The scope part of an LDAP URL contained invalid characters.

***GLOBUS\_URL\_ERROR\_BAD\_FILTER*** -12 The filter part of an LDAP URL contained invalid characters.

***GLOBUS\_URL\_ERROR\_OUT\_OF\_MEMORY*** -13 The library was unable to allocate memory to create the the [globus\\_url\\_t](#) contents.

***GLOBUS\_URL\_ERROR\_INTERNAL\_ERROR*** -14 Some unexpected error occurred parsing the URL.

#### **4.15.3.4 int globus\_url\_destroy ([globus\\_url\\_t](#) \* url)**

Destroy a [globus\\_url\\_t](#) structure.

This function frees all memory associated with a [globus\\_url\\_t](#) structure.

##### **Parameters:**

*url* The url structure to destroy

##### **Return values:**

***GLOBUS\_SUCCESS*** The URL was successfully destroyed.

#### **4.15.3.5 int globus\_url\_get\_scheme (const char \* url\_string, [globus\\_url\\_scheme\\_t](#) \* scheme\_type)**

Get the scheme of an URL.

This function determines the scheme type of the url string, and populates the variable pointed to by second parameter with that value. This performs a less expensive parsing than [globus\\_url\\_parse\(\)](#) and is suitable for applications which need only to choose a handler based on the URL scheme.

##### **Parameters:**

*url\_string* The string containing the URL.

*scheme\_type* A pointer to a [globus\\_url\\_scheme\\_t](#) which will be set to the scheme.

##### **Return values:**

***GLOBUS\_SUCCESS*** The URL scheme was recognized, and *scheme\_type* has been updated.

***GLOBUS\_URL\_ERROR\_BAD\_SCHEME*** The URL scheme was not recognized.

#### **4.15.3.6 int globus\_url\_copy ([globus\\_url\\_t](#) \* dst, const [globus\\_url\\_t](#) \* src)**

Create a copy of an URL structure.

This function copies the contents of a url structure into another.

##### **Parameters:**

*dst* The URL structure to be populated with a copy of the contents of *src*.

*src* The original URL.

##### **Return values:**

***GLOBUS\_SUCCESS*** The URL was successfully copied.

***GLOBUS\_URL\_ERROR\_NULL\_URL*** One of the URLs was **GLOBUS\_NULL**.

***GLOBUS\_URL\_ERROR\_OUT\_OF\_MEMORY***; The library was unable to allocate memory to create the the [globus\\_url\\_t](#) contents.

## 5 globus common Directory Documentation

### 5.1 /builddir/build/BUILD/globus\_common-11.5/library/ Directory Reference

library

#### Files

- file `closedir.c`
- file `freeaddrinfo.c`
- file `gai_strerror.c`
- file `getaddrinfo.c`
- file `getnameinfo.c`
- file `globus_args.c`
- file `globus_args.h`
- file `globus_callback.h`
- file `globus_callback_nothreads.c`
- file `globus_callback_threads.c`
- file `globus_common.c`
- file `globus_common.h`
- file `globus_common_include.h`
- file `globus_common_paths.c`
- file `globus_debug.c`
- file `globus_debug.h`
- file `globus_error.c`
- file `globus_error.h`
- file `globus_error_errno.c`
- file `globus_error_errno.h`
- file `globus_error_generic.c`
- file `globus_error_generic.h`
- file `globus_error_hierarchy.c`
- file `globus_error_hierarchy.h`
- file `globus_error_string.c`
- file `globus_error_string.h`
- file `globus_extension.c`
- file `globus_extension.h`
- file `globus_fifo.c`
- file `globus_fifo.h`
- file `globus_handle_table.c`
- file `globus_handle_table.h`
- file `globus_hashtable.c`
- file `globus_hashtable.h`
- file `globus_i_callback.h`
- file `globus_i_error_errno.c`
- file `globus_i_error_errno.h`
- file `globus_i_error_generic.c`
- file `globus_i_error_generic.h`
- file `globus_i_thread.h`
- file `globus_libc.c`

- file **globus\_libc.h**
- file **globus\_libc\_setenv.c**
- file **globus\_libtool\_windows.c**
- file **globus\_libtool\_windows.h**
- file **globus\_list.c**
- file **globus\_list.h**
- file **globus\_logging.c**
- file **globus\_logging.h**
- file **globus\_memory.c**
- file **globus\_memory.h**
- file **globus\_module.c**
- file **globus\_module.h**
- file **globus\_netos\_libc.h**
- file **globus\_object.c**
- file **globus\_object.h**
- file **globus\_object\_cache.c**
- file **globus\_object\_cache.h**
- file **globus\_object\_hierarchy.c**
- file **globus\_object\_hierarchy.h**
- file **globus\_options.c**
- file **globus\_options.h**
- file **globus\_print.c**
- file **globus\_print.h**
- file **globus\_priority\_q.c**
- file **globus\_priority\_q.h**
- file **globus\_range\_list.c**
- file **globus\_range\_list.h**
- file **globus\_release.h**
- file **globus\_states.c**
- file **globus\_states.h**
- file **globus\_strptime.c**
- file **globus\_strptime.h**
- file **globus\_symboltable.c**
- file **globus\_symboltable.h**
- file **globus\_thread\_common.c**
- file **globus\_thread\_common.h**
- file **globus\_thread\_external.c**
- file **globus\_thread\_external.h**
- file **globus\_thread\_none.c**
- file **globus\_thread\_none.h**
- file **globus\_thread\_pool.c**
- file **globus\_thread\_pool.h**
- file **globus\_thread\_pthreads.c**
- file **globus\_thread\_pthreads.h**
- file **globus\_thread\_rmutex.c**
- file **globus\_thread\_rmutex.h**
- file **globus\_thread\_rw\_mutex.c**
- file **globus\_thread\_rw\_mutex.h**
- file **globus\_thread\_solaristhreads.c**
- file **globus\_thread\_solaristhreads.h**
- file **globus\_thread\_sproc.c**
- file **globus\_thread\_sproc.h**

- file `globus_thread_windows.c`
- file `globus_thread_windows.h`
- file `globus_tilde_expand.c`
- file `globus_tilde_expand.h`
- file `globus_time.c`
- file `globus_time.h`
- file `globus_url.c`
- file `globus_url.h`
- file `globus_uuid.c`
- file `globus_uuid.h`
- file `inet_addr.c`
- file `inet_pton.c`
- file `opendir.c`
- file `readdir.c`
- file `rewinddir.c`

## 6 globus common Data Structure Documentation

### 6.1 globus\_url\_t Struct Reference

Parsed URLs.

#### Data Fields

- char \* [scheme](#)
- [globus\\_url\\_scheme\\_t](#) `scheme_type`
- char \* [user](#)
- char \* [password](#)
- char \* [host](#)
- unsigned short [port](#)
- char \* [url\\_path](#)
- char \* [dn](#)
- char \* [attributes](#)
- char \* [scope](#)
- char \* [filter](#)
- char \* [url\\_specific\\_part](#)

#### 6.1.1 Detailed Description

Parsed URLs.

This structure contains the fields which were parsed from an string representation of an URL. There are no methods to access fields of this structure.

#### 6.1.2 Field Documentation

##### 6.1.2.1 char\* [globus\\_url\\_t::scheme](#)

A string containing the URL's scheme (http, ftp, etc).

#### **6.1.2.2   `globus_url_scheme_t globus_url_t::scheme_type`**

An enumerated scheme type.

This is derived from the scheme string

#### **6.1.2.3   `char* globus_url_t::user`**

The username portion of the URL.

[ftp, gsiftp]

#### **6.1.2.4   `char* globus_url_t::password`**

The user's password from the URL.

[ftp, gsiftp]

#### **6.1.2.5   `char* globus_url_t::host`**

The host name or IP address of the URL.

[ftp, gsiftp, http, https, ldap, x-nexus]

#### **6.1.2.6   `unsigned short globus_url_t::port`**

The TCP port number of the service providing the URL [ftp, gsiftp, http, https, ldap, x-nexus].

#### **6.1.2.7   `char* globus_url_t::url_path`**

The path name of the resource on the service providing the URL.

[ftp, gsiftp, http, https]

#### **6.1.2.8   `char* globus_url_t::dn`**

The distinguished name for the base of an LDAP search.

[ldap]

#### **6.1.2.9   `char* globus_url_t::attributes`**

The list of attributes which should be returned from an LDAP search.

[ldap]

#### **6.1.2.10   `char* globus_url_t::scope`**

The scope of an LDAP search.

[ldap]

#### **6.1.2.11   `char* globus_url_t::filter`**

The filter to be applied to an LDAP search [ldap].

#### **6.1.2.12 char\* [globus\\_url\\_t::url\\_specific\\_part](#)**

An unparsed string containing the remaining text after the optional host and port of an unknown URL, or the contents of a x-gass-cache URL [x-gass-cache, unknown].

# Index

/builddir/build/BUILD/globus\_common-11.5/library/  
Directory Reference, [25](#)

attributes

globus\_url\_t, [28](#)

dn

globus\_url\_t, [28](#)

Error Construction, [8](#), [13](#)

Error Data Accessors and Modifiers, [10](#), [15](#)

Error Handling Helpers, [11](#), [18](#)

filter

globus\_url\_t, [28](#)

Globus Callback, [2](#)

Globus Callback API, [2](#)

Globus Callback Signal Handling, [6](#)

Globus Callback Spaces, [2](#)

Globus Errno Error API, [8](#)

Globus Error API, [12](#)

Globus Generic Error API, [13](#)

Globus Thread API, [20](#)

globus\_callback\_add\_wakeup\_handler

globus\_callback\_signal, [7](#)

GLOBUS\_CALLBACK\_GLOBAL\_SPACE

globus\_callback\_spaces, [3](#)

globus\_callback\_signal

globus\_callback\_add\_wakeup\_handler, [7](#)

globus\_callback\_space\_register\_signal\_handler, [7](#)

globus\_callback\_unregister\_signal\_handler, [7](#)

GLOBUS\_SIGNAL\_INTERRUPT, [7](#)

globus\_callback\_space\_attr\_destroy

globus\_callback\_spaces, [5](#)

globus\_callback\_space\_attr\_get\_behavior

globus\_callback\_spaces, [5](#)

globus\_callback\_space\_attr\_init

globus\_callback\_spaces, [4](#)

globus\_callback\_space\_attr\_set\_behavior

globus\_callback\_spaces, [5](#)

GLOBUS\_CALLBACK\_SPACE\_BEHAVIOR\_-  
SERIALIZED

globus\_callback\_spaces, [3](#)

GLOBUS\_CALLBACK\_SPACE\_BEHAVIOR\_-  
SINGLE

globus\_callback\_spaces, [3](#)

globus\_callback\_space\_behavior\_t

globus\_callback\_spaces, [3](#)

GLOBUS\_CALLBACK\_SPACE\_BEHAVIOR\_-  
THREADED

globus\_callback\_spaces, [3](#)

globus\_callback\_space\_destroy

globus\_callback\_spaces, [4](#)

globus\_callback\_space\_get

globus\_callback\_spaces, [5](#)

globus\_callback\_space\_get\_depth

globus\_callback\_spaces, [6](#)

globus\_callback\_space\_init

globus\_callback\_spaces, [3](#)

globus\_callback\_space\_is\_single

globus\_callback\_spaces, [6](#)

globus\_callback\_space\_reference

globus\_callback\_spaces, [4](#)

globus\_callback\_space\_register\_signal\_handler

globus\_callback\_signal, [7](#)

globus\_callback\_spaces

GLOBUS\_CALLBACK\_SPACE\_BEHAVIOR\_-  
SERIALIZED, [3](#)

GLOBUS\_CALLBACK\_SPACE\_BEHAVIOR\_-  
SINGLE, [3](#)

GLOBUS\_CALLBACK\_SPACE\_BEHAVIOR\_-  
THREADED, [3](#)

globus\_callback\_spaces

GLOBUS\_CALLBACK\_GLOBAL\_SPACE, [3](#)

globus\_callback\_space\_attr\_destroy, [5](#)

globus\_callback\_space\_attr\_get\_behavior, [5](#)

globus\_callback\_space\_attr\_init, [4](#)

globus\_callback\_space\_attr\_set\_behavior, [5](#)

globus\_callback\_space\_behavior\_t, [3](#)

globus\_callback\_space\_destroy, [4](#)

globus\_callback\_space\_get, [5](#)

globus\_callback\_space\_get\_depth, [6](#)

globus\_callback\_space\_init, [3](#)

globus\_callback\_space\_is\_single, [6](#)

globus\_callback\_space\_reference, [4](#)

globus\_callback\_unregister\_signal\_handler

globus\_callback\_signal, [7](#)

globus\_common\_thread

globus\_condattr\_getspace, [20](#)

globus\_condattr\_setspace, [20](#)

globus\_condattr\_getspace

globus\_common\_thread, [20](#)

globus\_condattr\_setspace

globus\_common\_thread, [20](#)

globus\_errno\_error\_accessor

globus\_error\_errno\_get\_errno, [10](#)

globus\_error\_errno\_set\_errno, [10](#)

globus\_errno\_error\_object

globus\_error\_construct\_errno\_error, [9](#)

globus\_error\_initialize\_errno\_error, [9](#)

GLOBUS\_ERROR\_TYPE\_ERRNO, [9](#)

globus\_errno\_error\_utility

globus\_error\_errno\_match, [11](#)

globus\_error\_wrap\_errno\_error, [11](#)

globus\_error\_construct\_errno\_error

globus\_errno\_error\_object, [9](#)

- globus\_error\_construct\_error
  - globus\_generic\_error\_object, [14](#)
- globus\_error\_errno\_get\_errno
  - globus\_errno\_error\_accessor, [10](#)
- globus\_error\_errno\_match
  - globus\_errno\_error\_utility, [11](#)
- globus\_error\_errno\_set\_errno
  - globus\_errno\_error\_accessor, [10](#)
- globus\_error\_get\_cause
  - globus\_generic\_error\_accessor, [17](#)
- globus\_error\_get\_long\_desc
  - globus\_generic\_error\_accessor, [18](#)
- globus\_error\_get\_short\_desc
  - globus\_generic\_error\_accessor, [17](#)
- globus\_error\_get\_source
  - globus\_generic\_error\_accessor, [16](#)
- globus\_error\_get\_type
  - globus\_generic\_error\_accessor, [17](#)
- globus\_error\_initialize\_errno\_error
  - globus\_errno\_error\_object, [9](#)
- globus\_error\_initialize\_error
  - globus\_generic\_error\_object, [15](#)
- globus\_error\_match
  - globus\_generic\_error\_utility, [19](#)
- globus\_error\_print\_chain
  - globus\_generic\_error\_utility, [19](#)
- globus\_error\_print\_friendly
  - globus\_generic\_error\_utility, [19](#)
- globus\_error\_set\_cause
  - globus\_generic\_error\_accessor, [17](#)
- globus\_error\_set\_long\_desc
  - globus\_generic\_error\_accessor, [18](#)
- globus\_error\_set\_short\_desc
  - globus\_generic\_error\_accessor, [18](#)
- globus\_error\_set\_source
  - globus\_generic\_error\_accessor, [16](#)
- globus\_error\_set\_type
  - globus\_generic\_error\_accessor, [17](#)
- GLOBUS\_ERROR\_TYPE\_ERRNO
  - globus\_errno\_error\_object, [9](#)
- GLOBUS\_ERROR\_TYPE\_GLOBUS
  - globus\_generic\_error\_object, [14](#)
- globus\_error\_v\_construct\_error
  - globus\_generic\_error\_object, [14](#)
- globus\_error\_wrap\_errno\_error
  - globus\_errno\_error\_utility, [11](#)
- globus\_generic\_error\_accessor
  - globus\_error\_get\_cause, [17](#)
  - globus\_error\_get\_long\_desc, [18](#)
  - globus\_error\_get\_short\_desc, [17](#)
  - globus\_error\_get\_source, [16](#)
  - globus\_error\_get\_type, [17](#)
  - globus\_error\_set\_cause, [17](#)
  - globus\_error\_set\_long\_desc, [18](#)
  - globus\_error\_set\_short\_desc, [18](#)
  - globus\_error\_set\_source, [16](#)
  - globus\_error\_set\_type, [17](#)
- globus\_error\_set\_type, [17](#)
- globus\_generic\_error\_object
  - globus\_error\_construct\_error, [14](#)
  - globus\_error\_initialize\_error, [15](#)
  - GLOBUS\_ERROR\_TYPE\_GLOBUS, [14](#)
  - globus\_error\_v\_construct\_error, [14](#)
- globus\_generic\_error\_utility
  - globus\_error\_match, [19](#)
  - globus\_error\_print\_chain, [19](#)
  - globus\_error\_print\_friendly, [19](#)
- GLOBUS\_SIGNAL\_INTERRUPT
  - globus\_callback\_signal, [7](#)
- globus\_url
  - GLOBUS\_URL\_NUM\_SCHEMES, [22](#)
  - GLOBUS\_URL\_SCHEME\_FILE, [22](#)
  - GLOBUS\_URL\_SCHEME\_FTP, [22](#)
  - GLOBUS\_URL\_SCHEME\_GSIFTP, [22](#)
  - GLOBUS\_URL\_SCHEME\_HTTP, [22](#)
  - GLOBUS\_URL\_SCHEME\_HTTPS, [22](#)
  - GLOBUS\_URL\_SCHEME\_LDAP, [22](#)
  - GLOBUS\_URL\_SCHEME\_UNKNOWN, [22](#)
  - GLOBUS\_URL\_SCHEME\_X\_GASS\_CACHE, [22](#)
  - GLOBUS\_URL\_SCHEME\_X\_NEXUS, [22](#)
- globus\_url
  - globus\_url\_copy, [24](#)
  - globus\_url\_destroy, [24](#)
  - globus\_url\_get\_scheme, [24](#)
  - globus\_url\_parse, [22](#)
  - globus\_url\_parse\_loose, [23](#)
  - globus\_url\_parse\_rfc1738, [22](#)
  - globus\_url\_scheme\_t, [21](#)
- globus\_url\_copy
  - globus\_url, [24](#)
- globus\_url\_destroy
  - globus\_url, [24](#)
- globus\_url\_get\_scheme
  - globus\_url, [24](#)
- GLOBUS\_URL\_NUM\_SCHEMES
  - globus\_url, [22](#)
- globus\_url\_parse
  - globus\_url, [22](#)
- globus\_url\_parse\_loose
  - globus\_url, [23](#)
- globus\_url\_parse\_rfc1738
  - globus\_url, [22](#)
- GLOBUS\_URL\_SCHEME\_FILE
  - globus\_url, [22](#)
- GLOBUS\_URL\_SCHEME\_FTP
  - globus\_url, [22](#)
- GLOBUS\_URL\_SCHEME\_GSIFTP
  - globus\_url, [22](#)
- GLOBUS\_URL\_SCHEME\_HTTP
  - globus\_url, [22](#)
- GLOBUS\_URL\_SCHEME\_HTTPS
  - globus\_url, [22](#)

- GLOBUS\_URL\_SCHEME\_LDAP
  - globus\_url, [22](#)
- globus\_url\_scheme\_t
  - globus\_url, [21](#)
- GLOBUS\_URL\_SCHEME\_UNKNOWN
  - globus\_url, [22](#)
- GLOBUS\_URL\_SCHEME\_X\_GASS\_CACHE
  - globus\_url, [22](#)
- GLOBUS\_URL\_SCHEME\_X\_NEXUS
  - globus\_url, [22](#)
- globus\_url\_t, [27](#)
  - attributes, [28](#)
  - dn, [28](#)
  - filter, [28](#)
  - host, [28](#)
  - password, [28](#)
  - port, [28](#)
  - scheme, [27](#)
  - scheme\_type, [27](#)
  - scope, [28](#)
  - url\_path, [28](#)
  - url\_specific\_part, [28](#)
  - user, [28](#)
- host
  - globus\_url\_t, [28](#)
- password
  - globus\_url\_t, [28](#)
- port
  - globus\_url\_t, [28](#)
- scheme
  - globus\_url\_t, [27](#)
- scheme\_type
  - globus\_url\_t, [27](#)
- scope
  - globus\_url\_t, [28](#)
- URL String Parser, [21](#)
- url\_path
  - globus\_url\_t, [28](#)
- url\_specific\_part
  - globus\_url\_t, [28](#)
- user
  - globus\_url\_t, [28](#)