

# globus common Reference Manual

## 10.2

Generated by Doxygen 1.2.18

Tue Aug 11 22:17:16 2009

## Contents

<a href="#">1 globus common Module Index</a>	<a href="#">1</a>
<a href="#">2 globus common Data Structure Index</a>	<a href="#">2</a>
<a href="#">3 globus common File Index</a>	<a href="#">2</a>
<a href="#">4 globus common Page Index</a>	<a href="#">2</a>
<a href="#">5 globus common Module Documentation</a>	<a href="#">2</a>
<a href="#">6 globus common Data Structure Documentation</a>	<a href="#">31</a>
<a href="#">7 globus common File Documentation</a>	<a href="#">33</a>
<a href="#">8 globus common Page Documentation</a>	<a href="#">36</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="#">4</a>
<a href="#">Globus Callback Spaces</a>	<a href="#">11</a>
<a href="#">Globus Callback Signal Handling</a>	<a href="#">15</a>
<a href="#">Globus Error API</a>	<a href="#">20</a>
<a href="#">Globus Errno Error API</a>	<a href="#">16</a>
<a href="#">Error Construction</a>	<a href="#">17</a>
<a href="#">Error Data Accessors and Modifiers</a>	<a href="#">18</a>
<a href="#">Error Handling Helpers</a>	<a href="#">19</a>
<a href="#">Globus Generic Error API</a>	<a href="#">20</a>
<a href="#">Error Construction</a>	<a href="#">21</a>
<a href="#">Error Data Accessors and Modifiers</a>	<a href="#">23</a>
<a href="#">Error Handling Helpers</a>	<a href="#">26</a>
<a href="#">Globus Thread API</a>	<a href="#">27</a>

URL String Parser 27

## 2 globus common Data Structure Index

### 2.1 globus common Data Structures

Here are the data structures with brief descriptions:

[globus.url\\_t](#) (Parsed URLs) 31

## 3 globus common File Index

### 3.1 globus common File List

Here is a list of all documented files with brief descriptions:

[globus.callback.h](#) (Globus Callback API) 33

[globus.thread.common.h](#) (Common Thread Interface) 35

## 4 globus common Page Index

### 4.1 globus common Related Pages

Here is a list of all related documentation pages:

Deprecated List 36

## 5 globus common Module Documentation

### 5.1 Globus Callback

Modules

[Globus Callback API](#)  
[Globus Callback Spaces](#)  
[Globus Callback Signal Handling](#)

Module Specific

```
#define GLOBUS_CALLBACK_MODULE
#define GLOBUS_POLL_MODULE
typedef int globus_callback_handle_t
typedef int globus_callback_space_t
typedef globus_callback_space_attr_t globus_callback_space_attr_t
```

```
enum globus_callback_error_type_t { GLOBUS_CALLBACK_ERROR_INVALID_CALLBACK_HANDLE =
1024, GLOBUS_CALLBACK_ERROR_INVALID_SPACE, GLOBUS_CALLBACK_ERROR_MEMORY_
ALLOC, GLOBUS_CALLBACK_ERROR_INVALID_ARGUMENT, GLOBUS_CALLBACK_ERROR_
ALREADY_CANCELED, GLOBUS_CALLBACK_ERROR_NO_ACTIVE_CALLBACK } g
```

### 5.1.1 Detailed Description

### 5.1.2 Define Documentation

#### 5.1.2.1 #define GLOBUS\_CALLBACK\_MODULE

Module descriptor for for globuscallback module.

Must be activated before any of the following api is called.

Note: You would not normally activate this module directly. Activating the GLOBUSCOMMON\_MODULE will in turn activate this also.

#### 5.1.2.2 #define GLOBUS\_POLL\_MODULE

Deprecated:

Backward compatible name

### 5.1.3 Typedef Documentation

#### 5.1.3.1 typedef int globus\_callback\_handle\_t

Handle for a periodic callback.

This handle can be copied or compared, and represented as NULL with GLOBUS\_HANDLE

#### 5.1.3.2 typedef int globus\_callback\_space\_t

Handle for a callback space.

This handle can be copied or compared and represented as NULL with GLOBUS\_HANDLE

#### 5.1.3.3 typedef struct globus\_callback\_spaceattr\_s globus\_callback\_spaceattr\_t

Handle for a space attr.

This handle can be copied and represented as NULL with GLOBUS\_HANDLE

### 5.1.4 Enumeration Type Documentation

#### 5.1.4.1 enum globus\_callback\_error\_type\_t

Possible error types returned by the api in this module.

You can use the error API to check results against these types.

See also:

[Error Handling Helpers](#)

Enumeration values:

GLOBUS\_CALLBACK\_ERROR\_INVALID\_CALLBACK\_HANDLE The callback handle is not valid or it has already been destroyed.  
 GLOBUS\_CALLBACK\_ERROR\_INVALID\_SPACE The space handle is not valid or it has already been destroyed.  
 GLOBUS\_CALLBACK\_ERROR\_MEMORY\_ALLOC Could not allocate memory for an internal structure.  
 GLOBUS\_CALLBACK\_ERROR\_INVALID\_ARGUMENT One of the arguments is NULL or out of range.  
 GLOBUS\_CALLBACK\_ERROR\_ALREADY\_CANCELED Attempt to unregister callback again.  
 GLOBUS\_CALLBACK\_ERROR\_NO\_ACTIVE\_CALLBACK Attempt to retrieve info about a callback not in callers's stack.

## 5.2 Globus Callback API

Convenience Macros

```
#define globuscallbackpoll(a)
#define globuspoll_blocking()
#define globuspoll_nonblocking()
#define globuspoll()
#define globussignalpoll()
#define globuscallbackregisteroneshot(callbackhandle, delaytime, callbackfunc, callbackuserarg)
#define globuscallbackregisterperiodic(callbackhandle, delaytime, period, callbackfunc, callbackuserarg)
#define globuscallbackregistersignalhandle(signum, persist, callbackfunc, callbackuserarg)
```

Callback Prototypes

```
typedef void( globuscallbackfunc_t )(void userarg)
```

Oneshot Callbacks

```
globusresult_t globuscallbackspaceregisteroneshot (globuscallbackhandle_t callbackhandle, const
globusreltime_t delaytime, globuscallbackfunc_t callbackfunc, void callbackuserarg, globuscallback-
space_t space)
```

Periodic Callbacks

```
globusresult_t globuscallbackspaceregisterperiodic (globuscallbackhandle_t callbackhandle, const
globusreltime_t delaytime, const globuseltime_t period, globuscallbackfunc_t callbackfunc, void
callbackuserarg, globuscallbackspace_t space)
globusresult_t globuscallbackunregister(globuscallbackhandle_t callbackhandle, globuscallbackfunc-
t unregistercallback, void unregarg, globusbool_t active)
globusresult_t globuscallbackadjustoneshot (globuscallbackhandle_t callbackhandle, const globus
reltime_t new_delay)
globusresult_t globuscallbackadjustperiod(globuscallbackhandle_t callbackhandle, const globuseltime_t
new_period)
```

Callback Polling

```
void globuscallbackspacepoll (const globusabstime_t timestop, globuscallbackspace_t space)
void globuscallbacksignalpoll ()
```

## Miscellaneous

```
globusbool_t globuscallbackgettimeout(globus_retime_t time_left)
globusbool_t globuscallbackhas_time_expired()
globusbool_t globuscallbackwas_restarted()
```

## 5.2.1 Detailed Description

## 5.2.2 Define Documentation

## 5.2.2.1 #define globuscallback\_poll(a)

Specifies the global space for [globuscallbackspacepoll\(\)](#).  
argument is the timeout

See also:

[globuscallbackspacepoll\(\)](#)

## 5.2.2.2 #define globuspoll\_blocking()

Specifies that [globuscallbackspacepoll\(\)](#) should poll on the global space with an infinite timeout.

See also:

[globuscallbackspacepoll\(\)](#)

## 5.2.2.3 #define globuspoll\_nonblocking()

Specifies that [globuscallbackspacepoll\(\)](#) should poll on the global space with an immediate timeout.

See also:

[globuscallbackspacepoll\(\)](#)

## 5.2.2.4 #define globuspoll()

Specifies that [globuscallbackspacepoll\(\)](#) should poll on the global space with an immediate timeout.

See also:

[globuscallbackspacepoll\(\)](#)

## 5.2.2.5 #define globussignalpoll()

Counterpart to [globuspoll\(\)](#).

See also:

[globuscallbacksignalpoll\(\)](#)

5.2.2.6 #define globuscallback\_register\_oneshot(callbackhandle, delaytime, callbackfunc, callback\_user\_arg)

Specifies the global space for [globuscallbackspaceregisteroneshot\(\)](#) all other arguments are the same as specified there.

See also:

[globuscallbackspaceregisteroneshot\(\)](#)

5.2.2.7 #define globuscallback\_register\_periodic(callback\_handle, delaytime, period, callbackfunc, callback\_user\_arg)

Specifies the global space for [globuscallbackspaceregisterperiodic\(\)](#) all other arguments are the same as specified there.

See also:

[globuscallbackspaceregisterperiodic\(\)](#)

5.2.2.8 #define globuscallback\_register\_signalhandler(signum, persist, callbackfunc, callback\_user\_arg)

Specifies the global space for [globuscallbackspaceregistersignalhandler\(\)](#) all other arguments are the same as specified there.

See also:

[globuscallbackspaceregistersignalhandler\(\)](#)

### 5.2.3 Typedef Documentation

5.2.3.1 typedef void( globus\_callback\_func\_t)( void user\_arg)

Globus callback prototype.

This is the signature of the function registered with the [globuscallbackregister](#) calls.

If this is a periodic callback, it is guaranteed that the call can NOT be reentered [globusthreadblocking.spacewill\\_block\(\)](#) is called (explicitly, or implicitly via [globuscondwait\(\)](#)). Also, if [globuscallbackunregister\(\)](#) is called to cancel this periodic from within this callback, it is guaranteed that the callback will NOT be requeued again.

If the function will block at all, the user should call [globuscallbackgettimeout\(\)](#) to see how long this function can safely block or call [globusthreadblocking.spacewill\\_block\(\)](#)

Parameters:

user\_arg The user argument registered with this callback

Returns:

void

See also:

[globuscallbackspaceregisteroneshot\(\)](#), [globuscallbackspaceregisterperiodic\(\)](#), [globusthreadblocking.spacewill\\_block\(\)](#), [globuscallbackgettimeout\(\)](#)

## 5.2.4 Function Documentation

5.2.4.1 `globusresult_t globus_callback_spaceregister_oneshot(globus\_callback\_handle\_t callback_handle, const globusreltime\_t delay_time, globus\_callback\_func\_t callback_func, void callback_user_arg, globus\_callback\_space\_t space)`

Register a oneshot some delay from now.

This function registers the `callback_func` to start some `delay_time` from now.

Parameters:

- `callback_handle` Storage for a handle. This may be NULL. If it is NOT NULL, you must unregister the callback to reclaim resources.
- `delay_time` The relative time from now to re this callback. If NULL, will re as soon as possible
- `callback_func` the user func to call
- `callback_user_arg` user arg that will be passed to callback
- `space` The space with which to register this callback

Returns:

GLOBUS\_CALLBACK\_ERROR\_INVALID\_ARGUMENT  
GLOBUS\_CALLBACK\_ERROR\_MEMORY\_ALLOC  
GLOBUS\_SUCCESS

See also:

[globus\\_callback\\_func\\_t](#) , [Globus Callback Spaces](#)

5.2.4.2 `globusresult_t globus_callback_spaceregister_periodic(globus\_callback\_handle\_t callback_handle, const globusreltime\_t delay_time, const globusreltime\_t period, globus\_callback\_func\_t callback_func, void callback_user_arg, globus\_callback\_space\_t space)`

Register a periodic callback.

This function registers a periodic `callback_func` to start some `delay_time` and run every `period` from then.

Parameters:

- `callback_handle` Storage for a handle. This may be NULL. If it is NOT NULL, you must cancel the periodic to reclaim resources.
- `delay_time` The relative time from now to re this callback. If NULL, will re the rst callback as soon as possible
- `period` The relative period of this callback
- `callback_func` the user func to call
- `callback_user_arg` user arg that will be passed to callback
- `space` The space with which to register this callback

Returns:

GLOBUS\_CALLBACK\_ERROR\_INVALID\_ARGUMENT  
GLOBUS\_CALLBACK\_ERROR\_MEMORY\_ALLOC  
GLOBUS\_SUCCESS

See also:

[globus\\_callback\\_unregister\(\)](#) , [globus\\_callback\\_func\\_t](#) , [Globus Callback Spaces](#)



5.2.4.3 `globusresult_t globus_callback_unregister (globus_callback_handle_t callback_handle, globus_callback_func_t unregister_callback, void unreg_arg, globus_bool_t active)`

Unregister a callback.

This function will cancel a callback and free the resources associated with the callback handle. If the callback was able to be canceled immediately (or if it has already run), `GLOBUS_SUCCESS` is returned and it is guaranteed that there are no running instances of the callback.

If the callback is currently running (or unstoppably about to be run), then the callback is prevented from being re-queued, but, the 'official' cancel is deferred until the last running instance of the callback returns. If you need to know when the callback is guaranteed to have been canceled, pass an `unregister_callback`.

If you would like to know if you unregistered a callback before it ran, pass storage for a boolean 'active'. This will be `GLOBUS_TRUE` if callback was running. `GLOBUS_FALSE` otherwise.

Parameters:

`callback_handle` the handle received from a `globus_callbackspace_register()` call

`unregister_callback` the function to call when the callback has been canceled and there are no running instances of it. This will be delivered to the same space used in the register call.

`unreg_arg` user arg that will be passed to the unregister callback

`active` storage for an indication of whether the callback was running when this call was made

Returns:

`GLOBUS_CALLBACK_ERROR_INVALID_CALLBACK_HANDLE`  
`GLOBUS_CALLBACK_ERROR_ALREADY_CANCELED`  
`GLOBUS_SUCCESS`

See also:

[globus\\_callbackspace\\_register\\_periodic\(\)](#), [globus\\_callback\\_func\\_t](#)

5.2.4.4 `globusresult_t globus_callback_adjust_oneshot (globus_callback_handle_t callback_handle, const globus_reftime_t new_delay)`

Adjust the delay of a oneshot callback.

This function allows a user to adjust the delay of a previously registered callback. It is safe to call this within or outside of the callback that is being modified.

Note if the oneshot has already been fired, this function will still return `GLOBUS_SUCCESS`, but won't affect anything.

Parameters:

`callback_handle` the handle received from [globus\\_callbackspace\\_register\\_oneshot\(\)](#) call

`new_delay` The new delay from now. If `NULL`, then callback will be fired as soon as possible.

Returns:

`GLOBUS_CALLBACK_ERROR_INVALID_CALLBACK_HANDLE`  
`GLOBUS_CALLBACK_ERROR_ALREADY_CANCELED`  
`GLOBUS_SUCCESS`

See also:

[globus\\_callbackspace\\_register\\_periodic\(\)](#)

5.2.4.5 `globusresult_t globus_callback_adjust_period (globus_callback_handle_t callback_handle, const globus_retime_t new_period)`

Adjust the period of a periodic callback.

This function allows a user to adjust the period of a previously registered callback. It is safe to call this within or outside of the callback that is being modified.

This func also allows a user to effectively 'suspend' a periodic callback until another time by passing a period of NULL. The callback can later be resumed by passing in a new period.

Note that the callback will not be red sooner than 'new\_period' from now. A 'suspended' callback must still be unregistered to free its resources.

Parameters:

`callback_handle` the handle received from `globus_callback_space_register_periodic()` call

`new_period` The new period. If NULL or `globus_retime_infinity`, then callback will be 'suspended' as soon as the last running instance of it returns.

Returns:

GLOBUS\_CALLBACK\_ERROR\_INVALID\_CALLBACK\_HANDLE  
GLOBUS\_CALLBACK\_ERROR\_ALREADY\_CANCELED  
GLOBUS\_SUCCESS

See also:

[globus\\_callback\\_space\\_register\\_periodic\(\)](#)

5.2.4.6 `void globus_callback_space_poll (const globus_abstime_t timestep, globus_callback_space_t space)`

Poll for ready callbacks.

This function is used to poll for registered callbacks.

For non-threaded builds, callbacks are not/can not be delivered unless this is called. Any call to this can cause callbacks registered with the 'global' space to be red. Whereas callbacks registered with a user's space will only be delivered when this is called with that space.

For threaded builds, this only needs to be called to poll user spaces with behavior == GLOBUS\_CALLBACK\_SPACE\_BEHAVIOR\_SINGLE. The 'global' space and other user spaces are constantly polled in a separate thread. (If it is called in a threaded build for these spaces, it will just yield its thread)

In general, you never need to call this function directly. It is called (when necessary) by `globus_cond_wait()`. The only case in which a user may wish to call this explicitly is if the application has no aspirations of ever being built threaded.

This function (when not yielding) will block up to timestep or until `globus_callback_signal_poll()` is called by one of the red callbacks. It will always try and kick out ready callbacks, regardless of the timestep.

Parameters:

`timestep` The time to block until. If this is NULL or less than the current time, an attempt to red only ready callbacks is made (no blocking).

`space` The callback space to poll. Note: regardless of what space is passed here, the 'global' space is also always polled.

Returns:

void

See also:

[Globus Callback Space](#), [globus\\_condattr\\_setspace\(\)](#)

## 5.2.4.7 void globuscallback\_signalpoll ()

Signal the poll.

This function signals [globuscallbackspacepoll\(\)](#) that something has changed and it should return to its caller as soon as possible.

In general, you never need to call this function directly. It is called (when necessary) by [globuscondsignal\(\)](#) or [globuscondbroadcast](#). The only case in which a user may wish to call this explicitly is if the application has no aspirations of ever being built threaded.

Returns:

void

See also:

[globuscallbackspacepoll\(\)](#)

## 5.2.4.8 globusbool\_t globus\_callback\_get\_timeout (globus\_retime\_t time\_left)

Get the amount of time left in a callback.

This function retrieves the remaining time a callback is allowed to run. If a callback has already timed out, `time_left` will be set to zero and `GLOBUS_TRUE` returned. This function is intended to be called within a callback's stack, but is harmless to call anywhere (will return `GLOBUS_FALSE` and an infinite `time_left`)

Parameters:

`time_left` storage for the remaining time.

Returns:

`GLOBUS_FALSE` if time remaining  
`GLOBUS_TRUE` if already timed out

## 5.2.4.9 globusbool\_t globus\_callback\_has\_time\_expired ()

See if there is remaining time in a callback.

This function returns `GLOBUS_TRUE` if the running time of a callback has already expired. This function is intended to be called within a callback's stack, but is harmless to call anywhere (will return `GLOBUS_FALSE`)

Returns:

`GLOBUS_FALSE` if time remaining  
`GLOBUS_TRUE` if already timed out

## 5.2.4.10 globusbool\_t globus\_callback\_was\_restarted ()

See if a callback has been restarted.

If the callback is a oneshot, this merely means the callback called [globusblocking\\_spacewill\\_block](#) (or [globuscondwait\(\)](#)) at some point.

For a periodic, it signifies the same and also that the periodic has been requeued. This means that the callback function may be reentered if the period is short enough (on a threaded build)

Returns:

`GLOBUS_FALSE` if not restarted  
`GLOBUS_TRUE` if restarted

## 5.3 Globus Callback Spaces

De nes

```
#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)
```

### 5.3.1 Detailed Description

### 5.3.2 De ne Documentation

#### 5.3.2.1 #define GLOBUS\_CALLBACK\_GLOBAL\_SPACE

The 'global' space handle.

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

### 5.3.3 Enumeration Type Documentation

#### 5.3.3.1 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)

Enumeration values:

`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.

### 5.3.4 Function Documentation

5.3.4.1 `globusresult_t globus_callback_spaceinit (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_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()`

5.3.4.2 `globusresult_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`

#### 5.3.4.3 globusresult\_t globus\_callback\_spacedestroy (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\(\)](#)

#### 5.3.4.4 globusresult\_t globus\_callback\_spaceattr\_init (globus\_callback\_spaceattr\_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\\_spaceattr\\_destroy\(\)](#)

Returns:

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

#### 5.3.4.5 globusresult\_t globus\_callback\_spaceattr\_destroy (globus\_callback\_spaceattr\_t attr)

Destroy a space attr.

Parameters:

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

Returns:

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

See also:

[globus\\_callback\\_spaceattr\\_init\(\)](#)

#### 5.3.4.6 globusresult\_t globus\_callback\_spaceattr\_set\_behavior (globus\_callback\_spaceattr\_t attr, globus\_callback\_spacebehavior\_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](#)

5.3.4.7 `globus_result_t globus_callback_spaceattr_get_behavior (globus_callback_spaceattr_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

5.3.4.8 `globus_result_t globus_callback_spaceget (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

5.3.4.9 `int globus_callback_spaceget_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

5.3.4.10 `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

## 5.4 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 callbackfunc, void callbackuserarg, globus\_callback\_space\_t space)`  
`globus_result_t globus\_callback\_unregister\_signal\_handler(int signum, globus\_callback\_func\_t unregister_callback, void unregarg)`  
`void globus\_callback\_add\_wakeup\_handler(void( wakeup)(void ), void userarg)`

### 5.4.1 Detailed Description

### 5.4.2 Define Documentation

#### 5.4.2.1 `#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.

### 5.4.3 Function Documentation

5.4.3.1 `globus_result_t globus_callback_space_register_signal_handler (int signum, globus_bool_t persist, globus\_callback\_func\_t callbackfunc, void callbackuserarg, 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.

`callbackfunc` 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

5.4.3.2 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

5.4.3.3 void globus\_callback\_add\_wakeup\_handler (void( wakeup)(void ), 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

## 5.5 Globus Errno Error API

Modules

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

### 5.5.1 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 object.

Any program that uses Globus Errno Error functions must include "globusmon.h".

## 5.6 Error Construction

### Construct Error

```
globusobject_t globus_error_construct_errno_error(globus_module_descriptor_t basesource, globusobject_t
basecause, const int system_errno)
```

### Initialize Error

```
globusobject_t globus_error_initialize_errno_error(globusobject_t error, globus_module_descriptor_t base-
source, globusobject_t basecause, const int system_errno)
```

### De nes

```
#define GLOBUS_ERROR_TYPE_ERRNO
```

### 5.6.1 Detailed Description

Create and initialize a Globus Errno Error object.

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

### 5.6.2 De ne Documentation

#### 5.6.2.1 #define GLOBUS\_ERROR\_TYPE\_ERRNO

Error type de nition.

### 5.6.3 Function Documentation

```
5.6.3.1 globusobject_t globus_error_construct_errno_error (globus_module_descriptor_t basesource
globusobject_t basecause, const int system_errno)
```

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

#### Parameters:

basesource Pointer to the originating module.

basecause 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.

5.6.3.2 `globusobject_t globus_error_initialize_errno_error (globus_object_t error, globus_module_descriptor_t basesource, globus_object_t basecause, const int system_errno)`

Initialize a previously allocated error of type `GLOBUS_ERRORTYPE_ERRNO`.

Parameters:

`error` The previously allocated error object.

`basesource` Pointer to the originating module.

`basecause` 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 `globuserror_put()` on this object before passing it on.

## 5.7 Error Data Accessors and Modifiers

Get Errno

`int globuserror_errno_get_errno(globusobject_t error)`

Set Errno

`void globuserror_errno_set_errno(globusobject_t error, const int system_errno)`

### 5.7.1 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.

### 5.7.2 Function Documentation

5.7.2.1 `int globuserror_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

5.7.2.2 `void globuserror_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

## 5.8 Error Handling Helpers

### Error Match

```
globus_bool_t globus_error_erno_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 basesource, int system_errno, int type, const char source_le, const char sourcefunc, int sourceline, const char shortdescformat,...)
```

#### 5.8.1 Detailed Description

Helper functions for dealing with Globus Errno Error objects.

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

#### 5.8.2 Function Documentation

5.8.2.1 `globus_bool_t globus_error_erno_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

5.8.2.2 `globus_object_t globus_error_wrap_errno_error (globus_module_descriptor_t basesource, int system_errno, int type, const char source_le, const char sourcefunc, int sourceline, const char shortdescformat, ...)`

Allocate and initialize an error of type `GLOBUS_ERROR_TYPE_GLOBUS` which contains a causal error of type `GLOBUS_ERROR_TYPE_ERRNO`.

##### Parameters:

`basesource` 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_le` Name of le. Use `__FILE__`

`sourcefunc` Name of function. Use `globus.func.name` and declare your func with `GlobusFuncName(func)`  
`sourceline` Line number. Use `LINE` ...  
`short.descformat` 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.

## 5.9 Globus Error API

Intended use:.

Modules

[Globus Errno Error API](#)  
[Globus Generic Error API](#)

### 5.9.1 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".

## 5.10 Globus Generic Error API

Modules

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

## 5.10.1 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 "globuscommon.h".

## 5.11 Error Construction

## Construct Error

```
globus_object_t globus_error_construct_error (globus_module_descriptor_t basesource, globus_object_t
basecause, int type, const char* source_line, const char* sourcefunc, int sourceline, const char* short-
descformat,...)
globus_object_t globus_error_v_construct_error (globus_module_descriptor_t basesource, globus_object_t
basecause, const int type, const char* source_line, const char* sourcefunc, int sourceline, const char* short-
descformat, va_list ap)
```

## Initialize Error

```
globus_object_t globus_error_initialize_error (globus_object_t error, globus_module_descriptor_t basesource,
globus_object_t basecause, int type, const char* source_line, const char* sourcefunc, int sourceline, const
char* shortdescformat, va_list ap)
```

## Defines

```
#define GLOBUS_ERROR_TYPE_GLOBUS
```

## 5.11.1 Detailed Description

Create and initialize a Globus Generic Error object.

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

## 5.11.2 Define Documentation

## 5.11.2.1 #define GLOBUS\_ERROR\_TYPE\_GLOBUS

Error type definition.

## 5.11.3 Function Documentation

5.11.3.1 globus\_object\_t globus\_error\_construct\_error (globus\_module\_descriptor\_t basesource, globus\_object\_t basecause, int type, const char\* source\_line, const char\* sourcefunc, int sourceline, const char\* shortdescformat, ...)

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

Parameters:

basesource Pointer to the originating module.

basecause 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 le Name of le. Use `__FILE__`.  
 sourcefunc Name of function. Use `globus_func_name` and declare your func with `GlobusFuncName`.  
 sourceline Line number. Use `__LINE__`.  
 short\_descformat 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 `globus_object_free()`.  
 A `globus_result_t` may be obtained by calling `globus_error_put()` on this object.

5.11.3.2 `globus_object_t globus_error_v_construct_error (globus_module_descriptor_t basesource, globus_object_t basecause, const int type, const char source le, const char sourcefunc, int sourceline, const char short_descformat, va_list ap)`

Allocate and initialize an error of type `GLOBUS_ERRORTYPE_GLOBUS`.

#### Parameters:

basesource Pointer to the originating module.  
 basecause 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 le Name of le. Use `__FILE__`.  
 sourcefunc Name of function. Use `globus_func_name` and declare your func with `GlobusFuncName`.  
 sourceline Line number. Use `__LINE__`.  
 short\_descformat 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 `globus_object_free()`.  
 A `globus_result_t` may be obtained by calling `globus_error_put()` on this object.

5.11.3.3 `globus_object_t globus_error_initialize_error (globus_object_t error, globus_module_descriptor_t basesource, globus_object_t basecause, int type, const char source le, const char sourcefunc, int sourceline, const char short_descformat, va_list ap)`

Initialize a previously allocated error of type `GLOBUS_ERRORTYPE_GLOBUS`.

#### Parameters:

error The previously allocated error object.  
 basesource Pointer to the originating module.  
 basecause 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.

`sourcefile` Name of file. Use `FILE__`  
`sourcefunc` Name of function. Use `globus.func.name` and declare your func with `GlobusFuncName(funcname)`  
`sourceline` Line number. Use `LINE__`  
`short_descformat` 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 `globuserror.put()` on this object before passing it on.

## 5.12 Error Data Accessors and Modifiers

Get Source

```
globus.module.descriptor_t globuserror_get_source(globus_object_t error)
```

Set Source

```
void globuserror_set_source(globus_object_t error, globus.module.descriptor_t sourcemodule)
```

Get Cause

```
globus_object_t globuserror_get_cause(globus_object_t error)
```

Set Cause

```
void globuserror_set_cause(globus_object_t error, globus_object_t causalerror)
```

Get Type

```
int globuserror_get_type(globus_object_t error)
```

Set Type

```
void globuserror_set_type(globus_object_t error, const int type)
```

Get Short Description

```
char globuserror_get_short_desc(globus_object_t error)
```

Set Short Description

```
void globuserror_set_short_desc(globus_object_t error, const char short_descformat,...)
```

Get Long Description

```
char globuserror_get_long_desc(globus_object_t error)
```



## Set Long Description

```
void globuserror.setlong_desc(globusobject_t error, const char long_descformat,...)
```

## 5.12.1 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.

## 5.12.2 Function Documentation

## 5.12.2.1 globusmodule\_descriptor\_t globus\_error\_get\_source (globusobject\_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.

## 5.12.2.2 void globuserror\_set\_source (globusobject\_t error, globusmodule\_descriptor\_t sourcemodule)

Set the originating module descriptor in a error object.

Parameters:

error The error object for which to set the causative error

sourcemodule The originating module descriptor

Returns:

void

## 5.12.2.3 globusobject\_t globus\_error\_get\_cause (globusobject\_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.

## 5.12.2.4 void globuserror\_set\_cause (globusobject\_t error, globusobject\_t causalError)

Set the causative error in a error object.

Parameters:

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

causalError The causative error.

Returns:

void

#### 5.12.2.5 `int globuserror_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

#### 5.12.2.6 `void globuserror_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

#### 5.12.2.7 `char globuserror_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

#### 5.12.2.8 `void globuserror_set_short_desc (globus_object_t error, const char short_descformat, ...)`

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

Parameters:

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

`short_descformat` Short format string giving a succinct description of the error. To be passed on to the user.

`...` Arguments for the format string.

Returns:

void

#### 5.12.2.9 `char globuserror_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

5.12.2.10 void globus\_error\_set\_long\_desc (globus\_object\_t error, const char long\_descformat, ...)

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

Parameters:

error The error object for which to set the description

long\_descformat Longer format string giving a more detailed explanation of the error.

Returns:

void

## 5.13 Error Handling Helpers

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)

### 5.13.1 Detailed Description

Helper functions for dealing with Globus Generic Error objects.

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

### 5.13.2 Function Documentation

5.13.2.1 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

### 5.13.2.2 char globus\_error\_print\_chain (globus\_object\_t error)

Return a string containing all printable errors found in a error object and it's causative error chain.

If the GLOBUS\_ERROR\_VERBOSE env is set, ie, 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.

### 5.13.2.3 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.

## 5.14 Globus Thread API

## 5.15 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\_parserfc1738(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)

```

### 5.15.1 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.

### 5.15.2 Enumeration Type Documentation

#### 5.15.2.1 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](#)

Enumeration values:

```

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.

```

### 5.15.3 Function Documentation

#### 5.15.3.1 int globus\_url\_parse (const char url\_string, globus\_url\_t url)

Parse a string containing a URL into [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 urlString 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 [globusurl.t](#) contents.  
 GLOBUS.URL.ERROR.INTERNAL\_ERROR -14 Some unexpected error occurred parsing the URL.

5.15.3.2 int globusurl\_parse\_rfc1738 (const char url\_string, [globusurl\\_t](#) url)

Parse a string containing a URL into [globusurl.t](#).

Parameters:

url\_string String to parse  
 url Pointer to [globusurl.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 urlString 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 [globusurl.t](#) contents.  
 GLOBUS.URL.ERROR.INTERNAL\_ERROR -14 Some unexpected error occurred parsing the URL.

### 5.15.3.3 int globusurl\_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.

### 5.15.3.4 int globusurl\_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.

### 5.15.3.5 int globusurl\_get\_scheme (const char url\_string, globus\_url\_scheme\_t schemetype)

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 [globusurl\\_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.

`schemetype` 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 `schemetype` has been updated.

`GLOBUS_URL_ERROR_BAD_SCHEME` The URL scheme was not recognized.

5.15.3.6 `int globusurl_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.

## 6 globus common Data Structure Documentation

### 6.1 globusurl\_t Struct Reference

Parsed URLs.

**Data Fields**

```

char  scheme
globus_url_scheme_t schemetype
char  user
char  password
char  host
unsigned short port
char  url_path
char  dn
char  attributes
char  scope
char  lter
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 globusurl\_t::scheme

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

#### 6.1.2.2 [globusurl\\_schemet](#) globusurl\_t::scheme.type

An enumerated scheme type.

This is derived from the scheme string

#### 6.1.2.3 char globusurl\_t::user

The username portion of the URL.

[ftp, gsiftp]

#### 6.1.2.4 char globusurl\_t::password

The user's password from the URL.

[ftp, gsiftp]

#### 6.1.2.5 char globusurl\_t::host

The host name or IP address of the URL.

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

#### 6.1.2.6 unsigned short globusurl\_t::port

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

#### 6.1.2.7 char globusurl\_t::url\_path

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

[ftp, gsiftp, http, https]

#### 6.1.2.8 char globusurl\_t::dn

The distinguished name for the base of an LDAP search.

[ldap]

#### 6.1.2.9 char globusurl\_t::attributes

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

[ldap]

#### 6.1.2.10 char globusurl\_t::scope

The scope of an LDAP search.

[ldap]

## 6.1.2.11 char globus\_url\_iter::iter

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

## 6.1.2.12 char globus\_url\_iter::url\_specic\_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].

## 7 globus common File Documentation

### 7.1 globuscallback.h File Reference

Globus Callback API.

#### Module Specific

```
#define GLOBUS_CALLBACK_MODULE
#define GLOBUS_POLL_MODULE
typedef int globus_callback_handle_t
typedef int globus_callback_space_t
typedef globus_callback_space_attr_s globus_callback_space_attr_t
enum globus_callback_error_type_t { GLOBUS_CALLBACK_ERROR_INVALID_CALLBACK_HANDLE =
1024, GLOBUS_CALLBACK_ERROR_INVALID_SPACE, GLOBUS_CALLBACK_ERROR_MEMORY_
ALLOC, GLOBUS_CALLBACK_ERROR_INVALID_ARGUMENT, GLOBUS_CALLBACK_ERROR_
ALREADY_CANCELED, GLOBUS_CALLBACK_ERROR_NO_ACTIVE_CALLBACK } g
```

#### Convenience Macros

```
#define globus_callback_poll(a)
#define globus_poll_blocking()
#define globus_poll_nonblocking()
#define globus_poll()
#define globus_signal_poll()
#define globus_callback_register_one_shot(callback_handle, delay_time, callback_func, callback_user_arg)
#define globus_callback_register_periodic(callback_handle, delay_time, period, callback_func, callback_user_arg)
#define globus_callback_register_signal_handle(signum, persist, callback_func, callback_user_arg)
```

#### Callback Prototypes

```
typedef void( globus_callback_func_t)(void userarg)
```

#### Oneshot Callbacks

```
globus_result_t globus_callback_space_register_one_shot(globus_callback_handle_t callback_handle, const
globus_reltime_t delay_time, globus_callback_func_t callback_func, void callback_user_arg, globus_callback_
space_t space)
```

## Periodic Callbacks

```

globusresult_t globuscallbackspaceregisterperiodic (globuscallbackhandle_t callbackhandle, const
globusreltime_t delaytime, const globusreltime_t period, globuscallbackfunc_t callbackfunc, void
callbackuserarg, globuscallbackspace_t space)
globusresult_t globuscallbackunregister(globuscallbackhandle_t callbackhandle, globuscallbackfunc-
t unregistercallback, void unregarg, globusbool_t active)
globusresult_t globuscallbackadjustoneshot (globuscallbackhandle_t callbackhandle, const globus
reltime_t new_delay)
globusresult_t globuscallbackadjustperiod(globuscallbackhandle_t callbackhandle, const globusreltime_t
new_period)

```

## Callback Polling

```

void globuscallbackspacepoll (const globusabstime_t timestop, globuscallbackspace_t space)
void globuscallbacksignalpoll ()

```

## Miscellaneous

```

globusbool_t globuscallbackgettimeout(globusreltime_t time_left)
globusbool_t globuscallbackhasexpired()
globusbool_t globuscallbackwasrestarted()

```

## Defines

```

#define GLOBUS_CALLBACK_GLOBAL_SPACE
#define GLOBUS_SIGNAL_INTERRUPT

```

## Enumerations

```

enum globuscallbackspacebehavior_t { GLOBUS_CALLBACK_SPACEBEHAVIOR_SINGLE,
GLOBUS_CALLBACK_SPACEBEHAVIOR_SERIALIZED, GLOBUS_CALLBACK_SPACEBEHAVIOR_-
THREADED }

```

## Functions

```

globusresult_t globuscallbackspaceinit (globuscallbackspace_t space, globuscallbackspaceattr_t attr)
globusresult_t globuscallbackspacereference(globuscallbackspace_t space)
globusresult_t globuscallbackspacedestroy(globuscallbackspace_t space)
globusresult_t globuscallbackspaceattr_init (globuscallbackspaceattr_t attr)
globusresult_t globuscallbackspaceattr_destroy(globuscallbackspaceattr_t attr)
globusresult_t globuscallbackspaceattr.setbehavior (globuscallbackspaceattr_t attr, globuscallback-
spacebehavior_t behavior)
globusresult_t globuscallbackspaceattr.getbehavior (globuscallbackspaceattr_t attr, globuscallback-
spacebehavior_t behavior)
globusresult_t globuscallbackspaceget(globuscallbackspace_t space)
int globuscallbackspaceget.depth(globuscallbackspace_t space)
globusbool_t globuscallbackspaceis_single(globuscallbackspace_t space)
globusresult_t globuscallbackspaceregistersignalhandler (int signum, globusbool_t persist, globus-
callbackfunc_t callbackfunc, void callbackuserarg, globuscallbackspace_t space)

```

```

globus_result_t globus_callback_unregister_signal_handler (int signum, globus_callbackfunc_t unregister-
callback, void unregarg)
void globus_callback_add_wakeup_handler(void( wakeup)(void ), void userarg)

```

### 7.1.1 Detailed Description

Globus Callback API.

Source:

/home/globdev/CVS/globus-packages/common/source/library/globus\_callback.h,v

Date:

2006/01/19 05:54:13

Revision:

1.12

Author:

mmlink

## 7.2 globusthread.common.h File Reference

Common Thread Interface.

### 7.2.1 Detailed Description

Common Thread Interface.

Source:

/home/globdev/CVS/globus-packages/common/source/library/globusthreadcommon.h,v

Date:

2006/01/19 05:54:14

Revision:

1.6

Author:

mmlink

### 7.2.2 De ne Documentation

#### 7.2.2.1 #de ne globusthread\_blocking\_callback\_push(f, u, i)

Value:

```

globus_thread_blocking_space_callback_push(
    (f), (u), GLOBUS_CALLBACK_GLOBAL_SPACE, (i))
\

```

## 8 globus common Page Documentation

### 8.1 Deprecated List

Global [GLOBUS\\_POLL\\_MODULE](#)

---

## Index

### attributes

globus.url\_t, [32](#)

### dn

globus.url\_t, [32](#)

### Error Construction, [17](#), [21](#)

### Error Data Accessors and Modifiers, [18](#), [23](#)

### Error Handling Helpers, [19](#), [26](#)

### iter

globus.url\_t, [32](#)

### Globus Callback, [2](#)

### Globus Callback API, [4](#)

### Globus Callback Signal Handling, [4](#), [5](#)

### Globus Callback Spaces, [1](#)

### Globus Errno Error API, [16](#)

### Globus Error API, [20](#)

### Globus Generic Error API, [20](#)

### Globus Thread API, [27](#)

### globus.callback

GLOBUS\_CALLBACK\_ERRORALREADY -  
CANCELED, [4](#)

GLOBUS\_CALLBACK\_ERRORINVALID -  
ARGUMENT, [4](#)

GLOBUS\_CALLBACK\_ERRORINVALID -  
CALLBACK\_HANDLE, [4](#)

GLOBUS\_CALLBACK\_ERRORINVALID -  
SPACE, [4](#)

GLOBUS\_CALLBACK\_ERRORMEMORY -  
ALLOC, [4](#)

GLOBUS\_CALLBACK\_ERRORNO -  
ACTIVE\_CALLBACK, [4](#)

### globus.callback

globus.callbackerror.type\_t, [3](#)

globus.callbackhandlet, [3](#)

GLOBUS\_CALLBACK\_MODULE, [3](#)

globus.callbackspaceattr\_t, [3](#)

globus.callbackspace\_t, [3](#)

GLOBUS\_POLL\_MODULE, [3](#)

### globus.callback.h, [33](#)

### globus.callback.add.wakeuphandler

globus.callbacksignal, [16](#)

### globus.callback.adjust.oneshot

globus.callbackapi, [8](#)

### globus.callback.adjust.period

globus.callbackapi, [8](#)

### globus.callbackapi

globus.callbackadjust.oneshot, [8](#)

globus.callbackadjust.period, [8](#)

globus.callbackfunc.t, [6](#)

globus.callback.get.timeout, [10](#)

globus.callback.has.time.expired, [10](#)

globus.callbackpoll, [5](#)

globus.callback.register.oneshot, [5](#)

globus.callback.register.periodic, [6](#)

globus.callback.register.signal.handler, [6](#)

globus.callbacksignalpoll, [9](#)

globus.callbackspacepoll, [9](#)

globus.callbackspace.register.oneshot, [7](#)

globus.callbackspace.register.periodic, [7](#)

globus.callback.unregister, [7](#)

globus.callback.was.restart, [10](#)

globus.poll, [5](#)

globus.poll\_blocking, [5](#)

globus.poll\_nonblocking, [5](#)

globus.signalpoll, [5](#)

GLOBUS\_CALLBACK\_ERRORALREADY -  
CANCELED

globus.callback, [4](#)

GLOBUS\_CALLBACK\_ERRORINVALID -  
ARGUMENT

globus.callback, [4](#)

GLOBUS\_CALLBACK\_ERRORINVALID -  
CALLBACK\_HANDLE

globus.callback, [4](#)

GLOBUS\_CALLBACK\_ERRORINVALID -  
SPACE

globus.callback, [4](#)

GLOBUS\_CALLBACK\_ERRORMEMORY -  
ALLOC

globus.callback, [4](#)

GLOBUS\_CALLBACK\_ERRORNO -  
ACTIVE\_CALLBACK

globus.callback, [4](#)

globus.callbackerror.type\_t

globus.callback, [3](#)

globus.callbackfunc.t

globus.callbackapi, [6](#)

globus.callback.get.timeout

globus.callbackapi, [10](#)

GLOBUS\_CALLBACK\_GLOBAL\_SPACE

globus.callbackspace, [1](#)

globus.callbackhandlet

globus.callback, [3](#)

globus.callback.has.time.expired

globus.callbackapi, [10](#)

GLOBUS\_CALLBACK\_MODULE

globus.callback, [3](#)

globus.callbackpoll

globus.callbackapi, [5](#)

- globus.callbackregisteroneshot
  - globus.callbackapi, [5](#)
- globus.callbackregisterperiodic
  - globus.callbackapi, [6](#)
- globus.callbackregistersignalhandler
  - globus.callbackapi, [6](#)
- globus.callbacksignal
  - globus.callbackadd.wakeuphandler, [16](#)
  - globus.callbacksaceregistersignalhandler, [15](#)
  - globus.callbackunregistersignalhandler, [16](#)
  - GLOBUS\_SIGNAL\_INTERRUPT, [15](#)
- globus.callbacksignalpoll
  - globus.callbackapi, [9](#)
- globus.callbacksaceattr.destroy
  - globus.callbacksaces, [13](#)
- globus.callbacksaceattr.get.behavior
  - globus.callbacksaces, [14](#)
- globus.callbacksaceattr.init
  - globus.callbacksaces, [13](#)
- globus.callbacksaceattr.set.behavior
  - globus.callbacksaces, [13](#)
- globus.callbacksaceattr.t
  - globus.callback, [3](#)
- GLOBUS\_CALLBACK\_SPACEBEHAVIOR\_-SERIALIZED
  - globus.callbacksaces, [12](#)
- GLOBUS\_CALLBACK\_SPACEBEHAVIOR\_-SINGLE
  - globus.callbacksaces, [12](#)
- globus.callbacksacebehaviort
  - globus.callbacksaces, [11](#)
- GLOBUS\_CALLBACK\_SPACEBEHAVIOR\_-THREADED
  - globus.callbacksaces, [12](#)
- globus.callbacksacedestroy
  - globus.callbacksaces, [12](#)
- globus.callbacksaceget
  - globus.callbacksaces, [14](#)
- globus.callbacksaceget.depth
  - globus.callbacksaces, [14](#)
- globus.callbacksaceinit
  - globus.callbacksaces, [12](#)
- globus.callbacksaceis.single
  - globus.callbacksaces, [14](#)
- globus.callbacksacepoll
  - globus.callbackapi, [9](#)
- globus.callbacksacereference
  - globus.callbacksaces, [12](#)
- globus.callbacksaceregisteroneshot
  - globus.callbackapi, [7](#)
- globus.callbacksaceregisterperiodic
  - globus.callbackapi, [7](#)
- globus.callbacksaceregistersignalhandler
  - globus.callbacksignal, [15](#)
- globus.callbacksacet
  - globus.callback, [3](#)
- globus.callbacksaces
  - GLOBUS\_CALLBACK\_SPACEBEHAVIOR\_-SERIALIZED, [12](#)
  - GLOBUS\_CALLBACK\_SPACEBEHAVIOR\_-SINGLE, [12](#)
  - GLOBUS\_CALLBACK\_SPACEBEHAVIOR\_-THREADED, [12](#)
- globus.callbacksaces
  - GLOBUS\_CALLBACK\_GLOBAL\_SPACE, [11](#)
  - globus.callbacksaceattr.destroy, [13](#)
  - globus.callbacksaceattr.get.behavior, [14](#)
  - globus.callbacksaceattr.init, [13](#)
  - globus.callbacksaceattr.set.behavior, [13](#)
  - globus.callbacksacebehaviort, [11](#)
  - globus.callbacksacedestroy, [12](#)
  - globus.callbacksaceget, [14](#)
  - globus.callbacksaceget.depth, [14](#)
  - globus.callbacksaceinit, [12](#)
  - globus.callbacksaceis.single, [14](#)
  - globus.callbacksacereference, [12](#)
- globus.callbackunregister
  - globus.callbackapi, [7](#)
- globus.callbackunregistersignalhandler
  - globus.callbacksignal, [16](#)
- globus.callbackwas.restart
  - globus.callbackapi, [10](#)
- globus.errno.error.accessor
  - globus.error.errno.get.errno, [18](#)
  - globus.error.errno.set.errno, [18](#)
- globus.errno.error.object
  - globus.error.constructerrno.error, [17](#)
  - globus.error.initialize\_errno.error, [17](#)
  - GLOBUS\_ERROR\_TYPE\_ERRNO, [17](#)
- globus.errno.error.utility
  - globus.error.errno.match, [19](#)
  - globus.error.wrap\_errno.error, [19](#)
- globus.error.constructerrno.error
  - globus.errno.error.object, [17](#)
- globus.error.constructerror
  - globus.genericerror.object, [21](#)
- globus.error.errno.get.errno
  - globus.errno.error.accessor, [18](#)
- globus.error.errno.match
  - globus.errno.error.utility, [19](#)
- globus.error.errno.set.errno
  - globus.errno.error.accessor, [18](#)
- globus.error.get.cause
  - globus.genericerror.accessor, [24](#)
- globus.error.get.long.desc

- globus.genericerror.accessor [25](#)
- globus.error.get.short.desc
  - globus.genericerror.accessor [25](#)
- globus.error.get.source
  - globus.genericerror.accessor [24](#)
- globus.error.get.type
  - globus.genericerror.accessor [24](#)
- globus.error.initialize.errno.error
  - globus.errno.error.object, [17](#)
- globus.error.initialize.error
  - globus.genericerror.object, [22](#)
- globus.error.match
  - globus.genericerror.utility, [26](#)
- globus.error.print.chain
  - globus.genericerror.utility, [26](#)
- globus.error.print.friendly
  - globus.genericerror.utility, [27](#)
- globus.error.set.cause
  - globus.genericerror.accessor [24](#)
- globus.error.set.long.desc
  - globus.genericerror.accessor [25](#)
- globus.error.set.short.desc
  - globus.genericerror.accessor [25](#)
- globus.error.set.source
  - globus.genericerror.accessor [24](#)
- globus.error.set.type
  - globus.genericerror.accessor [25](#)
- GLOBUS.ERRORTYPE.ERRNO
  - globus.errno.error.object, [17](#)
- GLOBUS.ERRORTYPE.GLOBUS
  - globus.genericerror.object, [21](#)
- globus.error.v.constructerror
  - globus.genericerror.object, [22](#)
- globus.error.wrap.errno.error
  - globus.errno.error.utility, [19](#)
- globus.genericerror.accessor
  - globus.error.get.cause [24](#)
  - globus.error.get.long.desc, [25](#)
  - globus.error.get.short.desc, [25](#)
  - globus.error.get.source [24](#)
  - globus.error.get.type, [24](#)
  - globus.error.set.cause [24](#)
  - globus.error.set.long.desc, [25](#)
  - globus.error.set.short.desc, [25](#)
  - globus.error.set.source [24](#)
  - globus.error.set.type, [25](#)
- globus.genericerror.object
  - globus.error.constructerror, [21](#)
  - globus.error.initialize.error, [22](#)
  - GLOBUS.ERRORTYPE.GLOBUS, [21](#)
  - globus.error.v.constructerror, [22](#)
- globus.genericerror.utility
  - globus.error.match, [26](#)
  - globus.error.print.chain, [26](#)
  - globus.error.print.friendly, [27](#)
- globus.poll
  - globus.callbackapi, [5](#)
- globus.poll\_blocking
  - globus.callbackapi, [5](#)
- GLOBUS.POLL MODULE
  - globus.callback, [3](#)
- globus.poll\_nonblocking
  - globus.callbackapi, [5](#)
- GLOBUS.SIGNAL INTERRUPT
  - globus.callbacksignal, [15](#)
- globus.signalpoll
  - globus.callbackapi, [5](#)
- globus.threadblocking.callbackpush
  - globus.threadcommon.h, [35](#)
- globus.threadcommon.h, [35](#)
  - globus.threadblocking.callbackpush, [35](#)
- globus.url
  - GLOBUS.URL.NUM.SCHEMES, [28](#)
  - GLOBUS.URL.SCHEME.FILE, [28](#)
  - GLOBUS.URL.SCHEME.FTP, [28](#)
  - GLOBUS.URL.SCHEME.GSIFTP, [28](#)
  - GLOBUS.URL.SCHEME.HTTP, [28](#)
  - GLOBUS.URL.SCHEME.HTTPS, [28](#)
  - GLOBUS.URL.SCHEME.LDAP, [28](#)
  - GLOBUS.URL.SCHEME.UNKNOWN, [28](#)
  - GLOBUS.URL.SCHEME.X.GASS.CACHE, [28](#)
  - GLOBUS.URL.SCHEME.X.NEXUS, [28](#)
- globus.url
  - globus.url.copy, [31](#)
  - globus.url.destroy, [30](#)
  - globus.url.get.scheme, [30](#)
  - globus.url.parse, [28](#)
  - globus.url.parseloose, [29](#)
  - globus.url.parserfc1738, [29](#)
  - globus.url.scheme, [28](#)
- globus.url.copy
  - globus.url, [31](#)
- globus.url.destroy
  - globus.url, [30](#)
- globus.url.get.scheme
  - globus.url, [30](#)
- GLOBUS.URL.NUM.SCHEMES
  - globus.url, [28](#)
- globus.url.parse
  - globus.url, [28](#)
- globus.url.parseloose
  - globus.url, [29](#)
- globus.url.parserfc1738
  - globus.url, [29](#)
- GLOBUS.URL.SCHEME.FILE



- globus.url, [28](#)
- GLOBUS.URL.SCHEME.FTP
  - globus.url, [28](#)
- GLOBUS.URL.SCHEME.GSIFTP
  - globus.url, [28](#)
- GLOBUS.URL.SCHEME.HTTP
  - globus.url, [28](#)
- GLOBUS.URL.SCHEME.HTTPS
  - globus.url, [28](#)
- GLOBUS.URL.SCHEME.LDAP
  - globus.url, [28](#)
- globus.url.scheme
  - globus.url, [28](#)
- GLOBUS.URL.SCHEME.UNKNOWN
  - globus.url, [28](#)
- GLOBUS.URL.SCHEMEX.GASS.CACHE
  - globus.url, [28](#)
- GLOBUS.URL.SCHEMEX.NEXUS
  - globus.url, [28](#)
- globus.url.t, [31](#)
  - attributes, [32](#)
  - dn, [32](#)
  - lter, [32](#)
  - host, [32](#)
  - password, [32](#)
  - port, [32](#)
  - scheme, [32](#)
  - schemetype, [32](#)
  - scope, [32](#)
  - url\_path, [32](#)
  - url\_speci c\_part, [33](#)
  - user, [32](#)
- host
  - globus.url.t, [32](#)
- password
  - globus.url.t, [32](#)
- port
  - globus.url.t, [32](#)
- scheme
  - globus.url.t, [32](#)
- schemetype
  - globus.url.t, [32](#)
- scope
  - globus.url.t, [32](#)
- URL String Parser, [27](#)
- url\_path
  - globus.url.t, [32](#)
- url\_speci c\_part
  - globus.url.t, [33](#)
- user
  - globus.url.t, [32](#)