

globus common Reference Manual

11.2

Generated by Doxygen 1.3.5

Thu Feb 25 09:35:59 2010

Contents

1 globus common Module Index	1
2 globus common Data Structure Index	2
3 globus common Page Index	2
4 globus common Module Documentation	2
5 globus common Data Structure Documentation	32
6 globus common Page Documentation	34

1 globus common Module Index

1.1 globus common Modules

Here is a list of all modules:

Globus Callback	2
Globus Callback API	4
Globus Callback Spaces	11
Globus Callback Signal Handling	15
Globus Error API	20
Globus Errno Error API	17
Error Construction	17
Error Data Accessors and Modifiers	18
Error Handling Helpers	19
Globus Generic Error API	21
Error Construction	21
Error Data Accessors and Modifiers	23
Error Handling Helpers	27
Globus Thread API	28
URL String Parser	28

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)

32

3 globus common Page Index

3.1 globus common Related Pages

Here is a list of all related documentation pages:

[Deprecated List](#)

34

4 globus common Module Documentation

4.1 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_ALLOCS,`
 - `GLOBUS_CALLBACK_ERROR_INVALID_ARGUMENT,`
 - `GLOBUS_CALLBACK_ERROR_ALREADY_CANCELED,`
 - `GLOBUS_CALLBACK_ERROR_NO_ACTIVE_CALLBACK,`

4.1.1 Detailed Description

4.1.2 Define Documentation

4.1.2.1 #define GLOBUS_CALLBACK_MODULE

Module descriptor for for globus_callback module.

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

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

4.1.2.2 #define GLOBUS_POLL_MODULE

[Deprecated](#)

Backward compatible name

4.1.3 Typedef Documentation

4.1.3.1 typedef int [globus_callback_handle_t](#)

Handle for a periodic callback.

This handle can be copied or compared, and represented as NULL with GLOBUS_NULL_HANDLE

4.1.3.2 typedef int [globus_callback_space_t](#)

Handle for a callback space.

This handle can be copied or compared and represented as NULL with GLOBUS_NULL_HANDLE

4.1.3.3 typedef struct globus_l_callback_space_attr_t [globus_callback_space_attr_t](#)

Handle for a space attr.

This handle can be copied and represented as NULL with GLOBUS_NULL

4.1.4 Enumeration Type Documentation

4.1.4.1 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:

[globus_generic_error_utility](#)

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_ALLOCCould 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_CANCELEDAttempt to unregister callback again.

GLOBUS_CALLBACK_ERROR_NO_ACTIVE_CALLBACKAttempt to retrieve info about a callback not in callers's stack.

4.2 Globus Callback API

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_oneshot(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_handler(signum, persist, callback_func, callback_user_arg)`

Callback Prototypes

- `typedef void(globus_callback_func)(void user_arg)`

Oneshot Callbacks

- `globus_result_t globus_callback_space_register_oneshot(globus_callback_handle_t callback_handle, const globus_reftime_t delay_time, globus_callback_func_t callback_func, void callback_user_arg, globus_callback_space_t space)`

Periodic Callbacks

- `globus_result_t globus_callback_space_register_periodic(globus_callback_handle_t callback_handle, const globus_reftime_t delay_time, const globus_reftime_period, globus_callback_func_t callback_func, void callback_user_arg, globus_callback_space_t space)`
- `globus_result_t globus_callback_unregister(globus_callback_handle_t callback_handle, globus_callback_func_t unregister_callback, void unreg_arg, globus_bool_t active)`
- `globus_result_t globus_callback_adjust_oneshot(globus_callback_handle_t callback_handle, const globus_reftime_t new_delay)`
- `globus_result_t globus_callback_adjust_periodic(globus_callback_handle_t callback_handle, const globus_reftime_t new_period)`

Callback Polling

- `void globus_callback_space_poll(const globus_abstime_t timestop, globus_callback_space_t space)`
- `void globus_callback_signal_poll()`

Miscellaneous

- `globus_bool_t globus_callback_get_timeo(globus_reftime_t time_left)`
- `globus_bool_t globus_callback_has_time_expired()`
- `globus_bool_t globus_callback_was_restarted()`

4.2.1 Detailed Description

4.2.2 Define Documentation

4.2.2.1 #define globus_callback_poll(a)

Specifies the global space for [globus_callback_space_poll\(\)](#)

argument is the timeout

See also:

[globus_callback_space_poll\(\)](#)

4.2.2.2 #define globus_poll_blocking()

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

See also:

[globus_callback_space_poll\(\)](#)

4.2.2.3 #define globus_poll_nonblocking()

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

See also:

[globus_callback_space_poll\(\)](#)

4.2.2.4 #define globus_poll()

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

See also:

[globus_callback_space_poll\(\)](#)

4.2.2.5 #define globus_signal_poll()

Counterpart to [globus_poll\(\)](#)

See also:

[globus_callback_signal_poll\(\)](#)

4.2.2.6 #define globus_callback_register_oneshot(callback_handle, delay_time, callback_func, callback_user_arg)

Specifies the global space for [globus_callback_space_register_oneshot\(\)](#). Other arguments are the same as specified there.

See also:

[globus_callback_space_register_oneshot\(\)](#)

4.2.2.7 #define globus_callback_register_periodic(callback_handle, delay_time, period, callback_func, callback_user_arg)

Specifies the global space for [globus_callback_space_register_periodic\(\)](#). Other arguments are the same as specified there.

See also:

[globus_callback_space_register_periodic\(\)](#)

4.2.2.8 #define globus_callback_register_signal_handler(signum, persist, callback_func, callback_user_arg)

Specifies the global space for [globus_callback_space_register_signal_handler\(\)](#). Other arguments are the same as specified there.

See also:

[globus_callback_space_register_signal_handler\(\)](#)

4.2.3 Typedef Documentation

4.2.3.1 typedef void([globus_callback_func_t](#))(void user_arg)

Globus callback prototype.

This is the signature of the function registered with the [globus_callback_register\(\)](#).

If this is a periodic callback, it is guaranteed that the call can NOT be reentered unless [globus_thread_blocking_space_will_block\(\)](#) is called (explicitly, or implicitly via [globus_cond_wait\(\)](#)). Also, [globus_callback_unregister\(\)](#) called to cancel this periodic from within this callback, it is guaranteed that the callback will NOT be requested again.

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

Parameters:

user_arg The user argument registered with this callback

Returns:

- void

See also:

[globus_callback_space_register_oneshot\(\)](#)
[globus_callback_space_register_periodic\(\)](#)
[globus_thread_blocking_space_will_block\(\)](#)
[globus_callback_get_timeout\(\)](#)

4.2.4 Function Documentation

4.2.4.1 globus_result_t globus_callback_space_register_oneshot([globus_callback_handle_t](#) callback_handle, const globus_reftime_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](#)

4.2.4.2 `globus_result_t globus_callback_space_register_periodic(globus_callback_handle_t callback_handle, const globus_retime_t delay_time, const globus_retime_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](#)

4.2.4.3 `globus_result_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_callback_space_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_callback_space_register_periodic\(\)](#)
[globus_callback_func_t](#)

4.2.4.4 `globus_result_t globus_callback_adjust_oneshot(globus_callback_handle_t callback_handle, const globus_retime_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_callback_space_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_callback_space_register_periodic\(\)](#)

4.2.4.5 `globus_result_t globus_callback_adjust_period(globus_callback_handle_t callback_handle, const globus_reftime_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 function 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 re-fired 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\(\)](#)

`new_period` The new period. If NULL or `globus_reftime_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\(\)](#)

4.2.4.6 `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 re-fired. 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 re-fired 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 re-fire 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_spaces](#)
[globus_condattr_setspace\(\)](#)

4.2.4.7 void globus_callback_signal_poll ()

Signal the poll.

This function signals [globus_callback_space_poll\(\)](#) 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 `globus_cond_signal()` or `globus_cond_broadcast`. 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:

[globus_callback_space_poll\(\)](#)

4.2.4.8 globus_bool_t globus_callback_get_timeout (globus_reftime_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

4.2.4.9 globus_bool_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

4.2.4.10 `globus_bool_t globus_callback_was_restarted()`

See if a callback has been restarted.

If the callback is a oneshot, this merely means the callback called `globus_thread_blocking_space_will_block` (or `globus_cond_wait()`) 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

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_in(globus_callback_space_t space, globus_callback_space_attr_t attr)`
- `globus_result_t globus_callback_space_reference(globus_callback_spacespace)`
- `globus_result_t globus_callback_space_destroy(globus_callback_spacespace)`
- `globus_result_t globus_callback_space_attr_init(globus_callback_space_attr attr)`
- `globus_result_t globus_callback_space_attr_destroy(globus_callback_space_attr attr)`
- `globus_result_t globus_callback_space_attr_set_behavior(globus_callback_space_attr attr, globus_callback_space_behavior behavior)`
- `globus_result_t globus_callback_space_attr_get_behavior(globus_callback_space_attr attr, globus_callback_space_behavior behavior)`
- `globus_result_t globus_callback_space_get(globus_callback_spacespace)`
- `int globus_callback_space_get_desc(globus_callback_spacespace)`
- `globus_bool_t globus_callback_space_is_single(globus_callback_spacespace)`

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)

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.

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 callback_func, void callback_user_arg, globus_callback_space_t space)
- globus_result_t globus_callback_unregister_signal_handler(int signum, globus_callback_func callback, void unreg_arg)
- void globus_callback_add_wakeup_handler(void (*wakeup)(void), void user_arg)

4.4.1 Detailed Description

4.4.2 De ne Documentation

4.4.2.1 #de ne 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 speci ed 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(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

4.5 Globus Errno Error API

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

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

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

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_err (globus_module_descriptor_t base_source, int system_errno, int type, const char* source_line, 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, 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

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

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

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_line, 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_line, 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_le, const char source_func, int source_line, const char short_desc_format, va_list ap)`

De nes

- `#define GLOBUS_ERROR_TYPE_GLOBUS`

4.11.1 Detailed Description

Create and initialize a Globus Generic Error object.

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

4.11.2 De ne Documentation

4.11.2.1 #define GLOBUS_ERROR_TYPE_GLOBUS

Error type de nition.

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_le, 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_le` Name of le. 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_le, 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_le` Name of le. 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, const int type, const char source_le, 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_le` Name of le. 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

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_terror, 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_terror)

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_terror, 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_terror)

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_terror, 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

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, 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.

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

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_loos`(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 `enumglobus_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_CACHEGASS Cache Entry.
- GLOBUS_URL_SCHEME_UNKNOWN Any other URL of the form ://.
- 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 [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 [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 Data Structure Documentation

5.1 `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 [lter](#)
- char [url_speci c_part](#)

5.1.1 Detailed Description

Parsed URLs.

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

5.1.2 Field Documentation

5.1.2.1 char [globus_url_t::scheme](#)

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

5.1.2.2 [globus_url_scheme_globus_url_t::scheme_type](#)

An enumerated scheme type.

This is derived from the scheme string

5.1.2.3 char [globus_url_t::user](#)

The username portion of the URL.

[ftp, gsiftp]

5.1.2.4 char [globus_url_t::password](#)

The user's password from the URL.

[ftp, gsiftp]

5.1.2.5 char [globus_url_t::host](#)

The host name or IP address of the URL.

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

5.1.2.6 unsigned short [globus_url_t::port](#)

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

5.1.2.7 char [globus_url_t::url_path](#)

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

[ftp, gsiftp, http, https]

5.1.2.8 char [globus_url_t::dn](#)

The distinguished name for the base of an LDAP search.

[ldap]

5.1.2.9 char [globus_url_t::attributes](#)

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

[ldap]

5.1.2.10 char [globus_url_t::scope](#)

The scope of an LDAP search.

[ldap]

5.1.2.11 char [globus_url_t::lter](#)

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

5.1.2.12 char [globus_url_t::url_speci c_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].

6 globus common Page Documentation

6.1 Deprecated List

Global [GLOBUS_POLL_MODULE](#)

Index

attributes

 globus_url_t, [33](#)

dn

 globus_url_t, [33](#)

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

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

Error Handling Helpers, [19](#), [27](#)

iter

 globus_url_t, [34](#)

Globus Callback, [2](#)

Globus Callback API, [4](#)

Globus Callback Signal Handling, [45](#)

Globus Callback Spaces, [1](#)

Globus Errno Error API, [17](#)

Globus Error API, [20](#)

Globus Generic Error API, [21](#)

Globus Thread API, [28](#)

globus_callback

 GLOBUS_CALLBACK_ERROR_ALREADY_-
 CANCELED, [3](#)

 GLOBUS_CALLBACK_ERROR_INVALID_-
 ARGUMENT, [3](#)

 GLOBUS_CALLBACK_ERROR_INVALID_-
 CALLBACK_HANDLE, [3](#)

 GLOBUS_CALLBACK_ERROR_INVALID_-
 SPACE, [3](#)

 GLOBUS_CALLBACK_ERROR_MEMORY_-
 ALLOC, [3](#)

 GLOBUS_CALLBACK_ERROR_NO_-
 ACTIVE_CALLBACK, [3](#)

globus_callback

 globus_callback_error_type_t, [3](#),

 globus_callback_handle_t, [3](#),

 GLOBUS_CALLBACK_MODULE, [2](#)

 globus_callback_space_attr_t, [3](#),

 globus_callback_space_t, [3](#),

 GLOBUS_POLL_MODULE, [3](#)

globus_callback_add_wakeup_handler

 globus_callback_signal_t, [16](#)

globus_callback_adjust_oneshot

 globus_callback_ap_t, [8](#)

globus_callback_adjust_period

 globus_callback_ap_t, [8](#)

globus_callback_api

 globus_callback_adjust_oneshot, [16](#),

 globus_callback_adjust_period, [16](#),

 globus_callback_func_t, [6](#)

 globus_callback_get_timeout, [10](#)

 globus_callback_has_time_expired, [10](#),

 globus_callback_poll_t, [5](#)

 globus_callback_register_oneshot, [16](#),

 globus_callback_register_periodic, [16](#),

 globus_callback_register_signal_handler, [16](#),

 globus_callback_signal_poll_t, [10](#)

 globus_callback_space_poll_t, [10](#),

 globus_callback_space_register_oneshot, [16](#),

 globus_callback_space_register_periodic, [16](#),

 globus_callback_unregister, [17](#),

 globus_callback_was_restarted, [10](#),

 globus_poll_t, [5](#)

 globus_poll_blocking_t, [5](#)

 globus_poll_nonblocking_t, [5](#)

 globus_signal_poll_t, [5](#)

GLOBUS_CALLBACK_ERROR_ALREADY_-
 CANCELED

 globus_callback_t, [3](#)

GLOBUS_CALLBACK_ERROR_INVALID_-
 ARGUMENT

 globus_callback_t, [3](#)

GLOBUS_CALLBACK_ERROR_INVALID_-
 CALLBACK_HANDLE

 globus_callback_t, [3](#)

GLOBUS_CALLBACK_ERROR_INVALID_SPACE
 globus_callback_t, [3](#)

GLOBUS_CALLBACK_ERROR_MEMORY_-
 ALLOC

 globus_callback_t, [3](#)

GLOBUS_CALLBACK_ERROR_NO_ACTIVE_-
 CALLBACK

 globus_callback_t, [3](#)

globus_callback_error_type_t

 globus_callback_t, [3](#)

globus_callback_func_t

 globus_callback_ap_t, [6](#)

globus_callback_get_timeout

 globus_callback_ap_t, [10](#)

GLOBUS_CALLBACK_GLOBAL_SPACE

 globus_callback_space_t, [1](#)

globus_callback_handle_t

 globus_callback_t, [3](#)

globus_callback_has_time_expired

 globus_callback_ap_t, [10](#)

GLOBUS_CALLBACK_MODULE

 globus_callback_t, [2](#)

globus_callback_poll

 globus_callback_ap_t, [5](#)

globus_callback_register_oneshot

globus_callback_ap5
 globus_callback_register_periodic
 globus_callback_ap5
 globus_callback_register_signal_handler
 globus_callback_ap6
 globus_callback_signal
 globus_callback_add_wakeup_handler16,
 globus_callback_space_register_signal_handler,
 16
 globus_callback_unregister_signal_handler16,
 GLOBUS_SIGNAL_INTERRUPT16
 globus_callback_signal_poll
 globus_callback_ap10
 globus_callback_space_attr_destroy
 globus_callback_spaces63
 globus_callback_space_attr_get_behavior
 globus_callback_spaces64
 globus_callback_space_attr_init
 globus_callback_spaces63
 globus_callback_space_attr_set_behavior
 globus_callback_spaces64
 globus_callback_space_attr_t
 globus_callback3
 GLOBUS_CALLBACK_SPACE_BEHAVIOR_-
 SERIALIZED
 globus_callback_spaces62
 GLOBUS_CALLBACK_SPACE_BEHAVIOR_-
 SINGLE
 globus_callback_spaces62
 globus_callback_space_behavior_t
 globus_callback_spaces62
 GLOBUS_CALLBACK_SPACE_BEHAVIOR_-
 THREADED
 globus_callback_spaces62
 globus_callback_space_destroy
 globus_callback_spaces63
 globus_callback_space_get
 globus_callback_spaces64
 globus_callback_space_get_depth
 globus_callback_spaces65
 globus_callback_space_init
 globus_callback_spaces62
 globus_callback_space_is_single
 globus_callback_spaces65
 globus_callback_space_poll
 globus_callback_ap9
 globus_callback_space_reference
 globus_callback_spaces62
 globus_callback_space_register_one_shot
 globus_callback_ap6
 globus_callback_space_register_periodic
 globus_callback_ap7,
 globus_callback_space_register_signal_handler
 globus_callback_signal16
 globus_callback_space_t
 globus_callback3
 globus_callback_spaces
 GLOBUS_CALLBACK_SPACE_BEHAVIOR_-
 SERIALIZED, 12
 GLOBUS_CALLBACK_SPACE_BEHAVIOR_-
 SINGLE, 12
 GLOBUS_CALLBACK_SPACE_BEHAVIOR_-
 THREADED, 12
 globus_callback_spaces
 GLOBUS_CALLBACK_GLOBAL_SPACE, 11
 globus_callback_space_attr_destroy63,
 globus_callback_space_attr_get_behavior16,
 globus_callback_space_attr_init13,
 globus_callback_space_attr_set_behavior16,
 globus_callback_space_behavior12,
 globus_callback_space_destroy63,
 globus_callback_space_get64,
 globus_callback_space_get_depth15,
 globus_callback_space_init12,
 globus_callback_space_is_single16,
 globus_callback_space_referent12,
 globus_callback_unregister
 globus_callback_ap7,
 globus_callback_unregister_signal_handler
 globus_callback_signal16
 globus_callback_was_restarted
 globus_callback_ap10
 globus_errno_error_accessor
 globus_error_errno_get_errno19
 globus_error_errno_set_errno19
 globus_errno_error_object
 globus_error_construct_errno_error68,
 globus_error_initialize_errno_error18
 GLOBUS_ERROR_TYPE_ERRNO18
 globus_errno_error_utility
 globus_error_errno_match20
 globus_error_wrap_errno_error20
 globus_error_construct_errno_error
 globus_errno_error_object18
 globus_error_construct_error
 globus_generic_error_object22
 globus_error_errno_get_errno
 globus_errno_error_accessor18,
 globus_error_errno_match
 globus_errno_error_utility20
 globus_error_errno_set_errno
 globus_errno_error_accessor18,
 globus_error_get_cause
 globus_generic_error_accessor25,
 globus_error_get_long_desc
 globus_generic_error_accessor26,

- globus_error_get_short_desc
 - globus_generic_error_access [26](#),
- globus_error_get_source
 - globus_generic_error_access [24](#),
- globus_error_get_type
 - globus_generic_error_access [25](#),
- globus_error_initialize_errno_error
 - globus_errno_error_object [18](#)
- globus_error_initialize_error
 - globus_generic_error_object [23](#)
- globus_error_match
 - globus_generic_error_utility [27](#)
- globus_error_print_chain
 - globus_generic_error_utility [27](#)
- globus_error_print_friendly
 - globus_generic_error_utility [27](#)
- globus_error_set_cause
 - globus_generic_error_access [25](#),
- globus_error_set_long_desc
 - globus_generic_error_access [26](#),
- globus_error_set_short_desc
 - globus_generic_error_access [26](#),
- globus_error_set_source
 - globus_generic_error_access [25](#),
- globus_error_set_type
 - globus_generic_error_access [25](#),
- GLOBUS_ERROR_TYPE_ERRNO
 - globus_errno_error_object [18](#)
- GLOBUS_ERROR_TYPE_GLOBUS
 - globus_generic_error_object [22](#)
- globus_error_v_construct_error
 - globus_generic_error_object [22](#)
- globus_error_wrap_errno_error
 - globus_errno_error_utility [20](#)
- globus_generic_error_accessor
 - globus_error_get_cause [25](#)
 - globus_error_get_long_desc [26](#)
 - globus_error_get_short_desc [26](#)
 - globus_error_get_source [24](#)
 - globus_error_get_type [25](#)
 - globus_error_set_cause [25](#)
 - globus_error_set_long_desc [26](#)
 - globus_error_set_short_desc [26](#)
 - globus_error_set_source [25](#)
 - globus_error_set_type [25](#)
- globus_generic_error_object
 - globus_error_construct_error [22](#)
 - globus_error_initialize_error [23](#)
 - GLOBUS_ERROR_TYPE_GLOBUS [22](#)
 - globus_error_v_construct_error [22](#)
- globus_generic_error_utility
 - globus_error_match [27](#)
 - globus_error_print_chain [27](#)
 - globus_error_print_friendly [27](#)
- globus_poll
 - globus_callback_ap [5](#)
- globus_poll_blocking
 - globus_callback_ap [5](#)
- GLOBUS_POLL_MODULE
 - globus_callback [3](#)
- globus_poll_nonblocking
 - globus_callback_ap [5](#)
- GLOBUS_SIGNAL_INTERRUPT
 - globus_callback_signal [16](#)
- globus_signal_poll
 - globus_callback_ap [5](#)
- globus_url
 - GLOBUS_URL_NUM_SCHEMES [29](#)
 - GLOBUS_URL_SCHEME_FILE [29](#)
 - GLOBUS_URL_SCHEME_FTP [29](#)
 - GLOBUS_URL_SCHEME_GSIFTP [29](#)
 - GLOBUS_URL_SCHEME_HTTP [29](#)
 - GLOBUS_URL_SCHEME_HTTPS [29](#)
 - GLOBUS_URL_SCHEME_LDAP [29](#)
 - GLOBUS_URL_SCHEME_UNKNOWN [29](#)
 - GLOBUS_URL_SCHEME_X_GASS_CACHE, [29](#)
 - GLOBUS_URL_SCHEME_X_NEXUS [29](#)
- globus_url
 - globus_url_copy [32](#)
 - globus_url_destroy [31](#)
 - globus_url_get_scheme [31](#)
 - globus_url_parse [29](#)
 - globus_url_parse_loose [30](#)
 - globus_url_parse_rfc1738 [30](#)
 - globus_url_scheme [29](#)
- globus_url_copy
 - globus_url [32](#)
- globus_url_destroy
 - globus_url [31](#)
- globus_url_get_scheme
 - globus_url [31](#)
- GLOBUS_URL_NUM_SCHEMES
 - globus_url [29](#)
- globus_url_parse
 - globus_url [29](#)
- globus_url_parse_loose
 - globus_url [30](#)
- globus_url_parse_rfc1738
 - globus_url [30](#)
- GLOBUS_URL_SCHEME_FILE
 - globus_url [29](#)
- GLOBUS_URL_SCHEME_FTP
 - globus_url [29](#)
- GLOBUS_URL_SCHEME_GSIFTP
 - globus_url [29](#)

- GLOBUS_URL_SCHEME_HTTP
 - globus_url, [29](#)
- GLOBUS_URL_SCHEME_HTTPS
 - globus_url, [29](#)
- GLOBUS_URL_SCHEME_LDAP
 - globus_url, [29](#)
- globus_url_scheme_t
 - globus_url, [29](#)
- GLOBUS_URL_SCHEME_UNKNOWN
 - globus_url, [29](#)
- GLOBUS_URL_SCHEME_X_GASS_CACHE
 - globus_url, [29](#)
- GLOBUS_URL_SCHEME_X_NEXUS
 - globus_url, [29](#)
- globus_url_t, [32](#)
 - attributes, [33](#)
 - dn, [33](#)
 - lter, [34](#)
 - host, [33](#)
 - password, [33](#)
 - port, [33](#)
 - scheme, [33](#)
 - scheme_type, [33](#)
 - scope, [34](#)
 - url_path, [33](#)
 - url_speci c_part, [34](#)
 - user, [33](#)
- host
 - globus_url_t, [33](#)
- password
 - globus_url_t, [33](#)
- port
 - globus_url_t, [33](#)
- scheme
 - globus_url_t, [33](#)
- scheme_type
 - globus_url_t, [33](#)
- scope
 - globus_url_t, [34](#)
- URL String Parser, [28](#)
- url_path
 - globus_url_t, [33](#)
- url_speci c_part
 - globus_url_t, [34](#)
- user
 - globus_url_t, [33](#)