

# globus callout Reference Manual

## 0.7

Generated by Doxygen 1.5.3

Fri Jun 26 20:38:55 2009

# Contents

<a href="#">1 Globus Callout API</a>	<a href="#">1</a>
<a href="#">2 globus callout Module Index</a>	<a href="#">1</a>
<a href="#">3 globus callout Module Documentation</a>	<a href="#">2</a>

## 1 Globus Callout API

This API is intended to ease integration of configurable callouts into the Globus Toolkit and to provide a platform independent way of dealing with runtime loadable functions. It (hopefully) achieves this goal by providing the following functionality:

- It provides a function for reading callout configuration files. Files are assumed to have the following format:
  - Anything after a '#' is assumed to be a comment
  - Blank lines are ignored
  - Lines specifying callouts have the format abstract type library symbol where "abstract type" denotes the type of callout, e.g. globus\_gram\_jobmanager\_authz, "library" denotes the library the callout can be found in and "symbol" denotes the function name of the callout.
- It provides a API function for registering callouts
- All callouts are assumed to have the function signature globus\_result\_t callout\_func(va\_list ap)
- It provides a function for calling a callout given a abstract type. If multiple callouts are defined for the same abstract type then all callouts for the abstract type will be called. Implementers should not rely on any correlation between the order of configuration and the order of invocation of callouts of the same abstract type.

Any program that uses Globus Callout functions must include "globus\_callout.h".

## 2 globus callout Module Index

### 2.1 globus callout Modules

Here is a list of all modules:

<b>Activation</b>	<a href="#">2</a>
<b>Callout Handle Operations</b>	<a href="#">2</a>
<b>Callout Configuration</b>	<a href="#">3</a>
<b>Callout Invocation</b>	<a href="#">4</a>
<b>Globus Callout Constants</b>	<a href="#">5</a>

## 3 globus callout Module Documentation

### 3.1 Activation

Globus Callout API uses standard Globus module activation and deactivation.

#### Defines

- `#define GLOBUS_CALLOUT_MODULE`

#### 3.1.1 Detailed Description

Globus Callout API uses standard Globus module activation and deactivation.

Before any Globus Callout API functions are called, the following function must be called:

```
globus_module_activate(GLOBUS_CALLOUT_MODULE)
```

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

To deactivate Globus Callout API, the following function must be called:

```
globus_module_deactivate(GLOBUS_CALLOUT_MODULE)
```

This function should be called once for each time Globus Callout API was activated.

#### 3.1.2 Define Documentation

##### 3.1.2.1 `#define GLOBUS_CALLOUT_MODULE`

Module descriptor.

## 3.2 Callout Handle Operations

Initialize and Destroy a Globus Callout Handle structure.

#### Initialize Handle

- `globus_result_t globus_callout_handle_init(globus_callout_handle_t *handle)`

#### Destroy Handle

- `globus_result_t globus_callout_handle_destroy(globus_callout_handle_t handle)`

#### Typedefs

- `typedef struct globus_i_callout_handle_s * globus_callout_handle_t`

### 3.2.1 Detailed Description

Initialize and Destroy a Globus Callout Handle structure.

This section defines operations for initializing and destroying Globus Callout Handle structure.

### 3.2.2 Typedef Documentation

#### 3.2.2.1 typedef struct globus\_i\_callout\_handle\_s\* globus\_callout\_handle\_t

Callout handle type definition.

### 3.2.3 Function Documentation

#### 3.2.3.1 globus\_result\_t globus\_callout\_handle\_init (globus\_callout\_handle\_t \* *handle*)

Initialize a Globus Callout Handle.

##### Parameters:

*handle* Pointer to the handle that is to be initialized

##### Returns:

GLOBUS\_SUCCESS if successful A Globus error object on failure: GLOBUS\_CALLOUT\_ERROR\_WITH\_HASHTABLE

#### 3.2.3.2 globus\_result\_t globus\_callout\_handle\_destroy (globus\_callout\_handle\_t *handle*)

Destroy a Globus Callout Handle.

##### Parameters:

*handle* The handle that is to be destroyed

##### Returns:

GLOBUS\_SUCCESS

## 3.3 Callout Configuration

Functions for registering callouts.

### Configure Callouts

- globus\_result\_t [globus\\_callout\\_read\\_config](#) (globus\_callout\_handle\_t handle, char \*filename)
- globus\_result\_t [globus\\_callout\\_register](#) (globus\_callout\_handle\_t handle, char \*type, char \*library, char \*symbol)

### 3.3.1 Detailed Description

Functions for registering callouts.

This section defines operations for registering callouts. Callouts may be registered either through a configuration file or through calls to `globus_callout_register`.

### 3.3.2 Function Documentation

#### 3.3.2.1 `globus_result_t globus_callout_read_config (globus_callout_handle_t handle, char *filename)`

Read callout configuration from file.

This function read a configuration file with the following format:

- Anything after a '#' is assumed to be a comment
- Blanks lines are ignored
- Lines specifying callouts have the format abstract type library symbol where "abstract type" denotes the type of callout, e.g. `globus_gram_jobmanager_authz`, "library" denotes the library the callout can be found in and "symbol" denotes the function name of the callout. The library argument can be specified in two forms, `libfoo` or `libfoo_<flavor>`. When using the former version the current flavor will automatically be added to the library name.

##### Parameters:

*handle* The handle that is to be configured

*filename* The file to read configuration from

##### Returns:

GLOBUS\_SUCCESS A Globus error object on failure: GLOBUS\_CALLOUT\_ERROR\_OPENING\_CONF\_FILE GLOBUS\_CALLOUT\_ERROR\_PARSING\_CONF\_FILE GLOBUS\_CALLOUT\_ERROR\_WITH\_HASHTABLE GLOBUS\_CALLOUT\_ERROR\_OUT\_OF\_MEMORY

#### 3.3.2.2 `globus_result_t globus_callout_register (globus_callout_handle_t handle, char *type, char *library, char *symbol)`

Register callout configuration.

This function registers a callout type in the given handle.

##### Parameters:

*handle* The handle that is to be configured

*type* The abstract type of the callout

*library* The location of the library containing the callout

*symbol* The symbol (ie function name) for the callout

##### Returns:

GLOBUS\_SUCCESS A Globus error object on failure: GLOBUS\_CALLOUT\_ERROR\_WITH\_HASHTABLE GLOBUS\_CALLOUT\_ERROR\_OUT\_OF\_MEMORY

### 3.4 Callout Invocation

Functions for invoking callouts.

#### Invoking Callouts

- `globus_result_t globus_callout_call_type (globus_callout_handle_t handle, char *type,...)`

## Typedefs

- typedef globus\_result\_t(\* globus\_callout\_function\_t)(va\_list ap)

### 3.4.1 Detailed Description

Functions for invoking callouts.

This section defines a operation for invoking callouts by their abstract type.

### 3.4.2 Typedef Documentation

#### 3.4.2.1 typedef globus\_result\_t(\* globus\_callout\_function\_t)(va\_list ap)

Callout function type definition.

### 3.4.3 Function Documentation

#### 3.4.3.1 globus\_result\_t globus\_callout\_call\_type (globus\_callout\_handle\_t *handle*, char \* *type*, ...)

Call a callout of specified abstract type.

This function looks up the callouts corresponding to the given type and invokes them with the passed arguments. If a invoked callout returns an error it will be chained to a error of the type GLOBUS\_CALLOUT\_ERROR\_CALLOUT\_ERROR and no more callouts will be called.

#### Parameters:

*handle* A configured callout handle

*type* The abstract type of the callout that is to be invoked

#### Returns:

GLOBUS\_SUCCESS A Globus error object on failure: GLOBUS\_CALLOUT\_ERROR\_TYPE\_NOT\_REGISTERED GLOBUS\_CALLOUT\_ERROR\_CALLOUT\_ERROR GLOBUS\_CALLOUT\_ERROR\_WITH\_DL GLOBUS\_CALLOUT\_ERROR\_WITH\_HASHTABLE GLOBUS\_CALLOUT\_ERROR\_OUT\_OF\_MEMORY

## 3.5 Globus Callout Constants

### Enumerations

- enum globus\_callout\_error\_t {  
GLOBUS\_CALLOUT\_ERROR\_SUCCESS = 0,  
GLOBUS\_CALLOUT\_ERROR\_WITH\_HASHTABLE = 1,  
GLOBUS\_CALLOUT\_ERROR\_OPENING\_CONF\_FILE = 2,  
GLOBUS\_CALLOUT\_ERROR\_PARSING\_CONF\_FILE = 3,  
GLOBUS\_CALLOUT\_ERROR\_WITH\_DL = 4,  
GLOBUS\_CALLOUT\_ERROR\_OUT\_OF\_MEMORY = 5,  
GLOBUS\_CALLOUT\_ERROR\_TYPE\_NOT\_REGISTERED = 6,  
GLOBUS\_CALLOUT\_ERROR\_CALLOUT\_ERROR = 7,  
GLOBUS\_CALLOUT\_ERROR\_LAST = 8 }

### 3.5.1 Enumeration Type Documentation

#### 3.5.1.1 enum globus\_callout\_error\_t

Globus Callout Error codes.

**Enumerator:**

***GLOBUS\_CALLOUT\_ERROR\_SUCCESS*** Success - never used.

***GLOBUS\_CALLOUT\_ERROR\_WITH\_HASHTABLE*** Hash table operation failed.

***GLOBUS\_CALLOUT\_ERROR\_OPENING\_CONF\_FILE*** Failed to open configuration file.

***GLOBUS\_CALLOUT\_ERROR\_PARSING\_CONF\_FILE*** Failed to parse configuration file.

***GLOBUS\_CALLOUT\_ERROR\_WITH\_DL*** Dynamic library operation failed.

***GLOBUS\_CALLOUT\_ERROR\_OUT\_OF\_MEMORY*** Out of memory.

***GLOBUS\_CALLOUT\_ERROR\_TYPE\_NOT\_REGISTERED*** The abstract type could not be found.

***GLOBUS\_CALLOUT\_ERROR\_CALLOUT\_ERROR*** The callout itself returned a error.

***GLOBUS\_CALLOUT\_ERROR\_LAST*** Last marker - never used.

# Index

Activation, [1](#)

Callout Configuration, [3](#)

Callout Handle Operations, [2](#)

Callout Invocation, [4](#)

Globus Callout Constants, [5](#)

globus\_callout\_activation

[GLOBUS\\_CALLOUT\\_MODULE](#), [2](#)

globus\_callout\_call

[globus\\_callout\\_call\\_type](#), [5](#)

[globus\\_callout\\_function\\_t](#), [5](#)

globus\_callout\_call\_type

[globus\\_callout\\_call](#), [5](#)

globus\_callout\_config

[globus\\_callout\\_read\\_config](#), [3](#)

[globus\\_callout\\_register](#), [4](#)

globus\_callout\_constants

[GLOBUS\\_CALLOUT\\_ERROR\\_CALLOUT\\_ERROR](#), [6](#)

[GLOBUS\\_CALLOUT\\_ERROR\\_LAST](#), [6](#)

[GLOBUS\\_CALLOUT\\_ERROR\\_OPENING\\_CONF\\_FILE](#), [5](#)

[GLOBUS\\_CALLOUT\\_ERROR\\_OUT\\_OF\\_MEMORY](#), [6](#)

[GLOBUS\\_CALLOUT\\_ERROR\\_PARSING\\_CONF\\_FILE](#), [6](#)

[GLOBUS\\_CALLOUT\\_ERROR\\_SUCCESS](#), [5](#)

[GLOBUS\\_CALLOUT\\_ERROR\\_TYPE\\_NOT\\_REGISTERED](#), [6](#)

[GLOBUS\\_CALLOUT\\_ERROR\\_WITH\\_DL](#), [6](#)

[GLOBUS\\_CALLOUT\\_ERROR\\_WITH\\_HASHTABLE](#), [5](#)

globus\_callout\_constants

[globus\\_callout\\_error\\_t](#), [5](#)

[GLOBUS\\_CALLOUT\\_ERROR\\_CALLOUT\\_ERROR](#)

[globus\\_callout\\_constants](#), [6](#)

[GLOBUS\\_CALLOUT\\_ERROR\\_LAST](#)

[globus\\_callout\\_constants](#), [6](#)

[GLOBUS\\_CALLOUT\\_ERROR\\_OPENING\\_CONF\\_FILE](#)

[globus\\_callout\\_constants](#), [5](#)

[GLOBUS\\_CALLOUT\\_ERROR\\_OUT\\_OF\\_MEMORY](#)

[globus\\_callout\\_constants](#), [6](#)

[GLOBUS\\_CALLOUT\\_ERROR\\_PARSING\\_CONF\\_FILE](#)

[globus\\_callout\\_constants](#), [6](#)

[GLOBUS\\_CALLOUT\\_ERROR\\_SUCCESS](#)

[globus\\_callout\\_constants](#), [5](#)

[globus\\_callout\\_error\\_t](#)

[globus\\_callout\\_constants](#), [5](#)

[GLOBUS\\_CALLOUT\\_ERROR\\_TYPE\\_NOT\\_REGISTERED](#)

[globus\\_callout\\_constants](#), [6](#)

[GLOBUS\\_CALLOUT\\_ERROR\\_WITH\\_DL](#)

[globus\\_callout\\_constants](#), [6](#)

[GLOBUS\\_CALLOUT\\_ERROR\\_WITH\\_HASHTABLE](#)

[globus\\_callout\\_constants](#), [5](#)

[globus\\_callout\\_function\\_t](#)

[globus\\_callout\\_call](#), [5](#)

[globus\\_callout\\_handle](#)

[globus\\_callout\\_handle\\_destroy](#), [3](#)

[globus\\_callout\\_handle\\_init](#), [3](#)

[globus\\_callout\\_handle\\_t](#), [2](#)

[globus\\_callout\\_handle\\_destroy](#)

[globus\\_callout\\_handle](#), [3](#)

[globus\\_callout\\_handle\\_init](#)

[globus\\_callout\\_handle](#), [3](#)

[globus\\_callout\\_handle\\_t](#)

[globus\\_callout\\_handle](#), [2](#)

[GLOBUS\\_CALLOUT\\_MODULE](#)

[globus\\_callout\\_activation](#), [2](#)

[globus\\_callout\\_read\\_config](#)

[globus\\_callout\\_config](#), [3](#)

[globus\\_callout\\_register](#)

[globus\\_callout\\_config](#), [4](#)