

# globus ftp control Reference Manual

## 2.10

Generated by Doxygen 1.3.6

Fri Jun 26 16:56:32 2009

# Contents

<b>1</b>	<b><a href="#">Globus GSIFTP Control Connection API</a></b>	<b>1</b>
<b>2</b>	<b><a href="#">globus ftp control Data Structure Index</a></b>	<b>1</b>
<b>3</b>	<b><a href="#">globus ftp control File Index</a></b>	<b>2</b>
<b>4</b>	<b><a href="#">globus ftp control Page Index</a></b>	<b>2</b>
<b>5</b>	<b><a href="#">globus ftp control Data Structure Documentation</a></b>	<b>2</b>
<b>6</b>	<b><a href="#">globus ftp control File Documentation</a></b>	<b>4</b>
<b>7</b>	<b><a href="#">globus ftp control Page Documentation</a></b>	<b>34</b>

## 1 Globus GSIFTP Control Connection API

The `globus_ftp_control` library provides low-level services needed to implement FTP client and servers. The API provided is protocol specific. See the GASS Transfer library for a protocol-independent transfer interface.

This data transfer portion of this API provides support for the standard data methods described in the [FTP Specification](#) as well as [extensions](#) for parallel, striped, and partial data transfer.

Any program that uses the GSIFTP Control Library must include "globus\_ftp\_control.h".

## 2 globus ftp control Data Structure Index

### 2.1 globus ftp control Data Structures

Here are the data structures with brief descriptions:

<a href="#">globus_ftp_control_auth_info_s</a> (Authentication Values )	2
<a href="#">globus_ftp_control_dcau_subject_s</a> (Control dcau subject authentication type )	2
<a href="#">globus_ftp_control_dcau_u</a> (Control dcau union )	2
<a href="#">globus_ftp_control_layout_u</a> (Control striping attribute union )	3
<a href="#">globus_ftp_control_parallelism_u</a> (Control parallelism attribute structure )	3
<a href="#">globus_ftp_control_round_robin_s</a> (Control striping round robin attribute structure )	3
<a href="#">globus_ftp_control_tcpbuffer_automatic_s</a> (Automatically set the TCP buffer/window size )	3
<a href="#">globus_ftp_control_tcpbuffer_default_t</a> (Don't change the TCP buffer/window size from the system default )	3
<a href="#">globus_ftp_control_tcpbuffer_fixed_t</a> (Set the TCP buffer/window size to a fixed value )	3
<a href="#">globus_ftp_control_tcpbuffer_t</a> (Control tcpbuffer attribute structure )	4

## 3 globus ftp control File Index

### 3.1 globus ftp control File List

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

<a href="#">globus_ftp_control.c</a> (FTP Control API Activation/Deactivation and Global State )	4
<a href="#">globus_ftp_control.h</a> (GSIFTP Control Connection API (Data structures and types) )	4
<a href="#">globus_ftp_control_client.c</a> (Client-side FTP Control API )	14
<a href="#">globus_ftp_control_data.c</a> (FTP Data Connection Configuration and Management )	21
<a href="#">globus_ftp_control_layout.c</a>	30
<a href="#">globus_ftp_control_server.c</a> (FTP Server-side Control Connection Management )	30

## 4 globus ftp control Page Index

### 4.1 globus ftp control Related Pages

Here is a list of all related documentation pages:

<a href="#">GridFTP: Protocol Extensions to FTP for the Grid</a>	34
--	----

## 5 globus ftp control Data Structure Documentation

### 5.1 globus\_ftp\_control\_auth\_info\_s Struct Reference

Authentication Values.

#### 5.1.1 Detailed Description

Authentication Values.

This structure is populated and passed back to the user via the [globus\\_ftp\\_control\\_auth\\_callback\\_t\(\)](#). It contains the information needed to decide if a client may use the server.

### 5.2 globus\_ftp\_control\_dcdu\_subject\_s Struct Reference

control dcdu subject authentication type

#### 5.2.1 Detailed Description

control dcdu subject authentication type

### 5.3 globus\_ftp\_control\_dcdu\_u Union Reference

control dcdu union

### **5.3.1 Detailed Description**

control dcu union

## **5.4 globus\_ftp\_control\_layout\_u Union Reference**

control striping attribute union

### **5.4.1 Detailed Description**

control striping attribute union

## **5.5 globus\_ftp\_control\_parallelism\_u Union Reference**

control parallelism attribute structure

### **5.5.1 Detailed Description**

control parallelism attribute structure

## **5.6 globus\_ftp\_control\_round\_robin\_s Struct Reference**

control striping round robin attribute structure

### **5.6.1 Detailed Description**

control striping round robin attribute structure

## **5.7 globus\_ftp\_control\_tcpbuffer\_automatic\_s Struct Reference**

Automatically set the TCP buffer/window size.

### **5.7.1 Detailed Description**

Automatically set the TCP buffer/window size.

## **5.8 globus\_ftp\_control\_tcpbuffer\_default\_t Struct Reference**

Don't change the TCP buffer/window size from the system default.

### **5.8.1 Detailed Description**

Don't change the TCP buffer/window size from the system default.

## **5.9 globus\_ftp\_control\_tcpbuffer\_fixed\_t Struct Reference**

Set the TCP buffer/window size to a fixed value.

### 5.9.1 Detailed Description

Set the TCP buffer/window size to a fixed value.

## 5.10 globus\_ftp\_control\_tcpbuffer\_t Union Reference

control tcpbuffer attribute structure

### 5.10.1 Detailed Description

control tcpbuffer attribute structure

## 6 globus ftp control File Documentation

### 6.1 globus\_ftp\_control.c File Reference

FTP Control API Activation/Deactivation and Global State.

#### Variables

- int [globus\\_i\\_ftp\\_control\\_debug\\_level](#) = 0

#### 6.1.1 Detailed Description

FTP Control API Activation/Deactivation and Global State.

#### 6.1.2 Variable Documentation

##### 6.1.2.1 int [globus\\_i\\_ftp\\_control\\_debug\\_level](#) = 0

Debugging level.

1 thru 3 enable debug output for control channel 4 thru 6 enable debug output for control and data channel

### 6.2 globus\_ftp\_control.h File Reference

GSIFTP Control Connection API (Data structures and types).

#### Data Structures

- struct [globus\\_ftp\\_control\\_dcau\\_subject\\_s](#)  
*control dcau subject authentication type*
- struct [globus\\_ftp\\_control\\_round\\_robin\\_s](#)  
*control striping round robin attribute structure*
- union [globus\\_ftp\\_control\\_dcau\\_u](#)  
*control dcau union*

- union [globus\\_ftp\\_control\\_layout\\_u](#)  
*control striping attribute union*
- union [globus\\_ftp\\_control\\_parallelism\\_u](#)  
*control parallelism attribute structure*
- struct [globus\\_ftp\\_control\\_tcpbuffer\\_default\\_t](#)  
*Don't change the TCP buffer/window size from the system default.*
- struct [globus\\_ftp\\_control\\_tcpbuffer\\_fixed\\_t](#)  
*Set the TCP buffer/window size to a fixed value.*
- struct [globus\\_ftp\\_control\\_tcpbuffer\\_automatic\\_s](#)  
*Automatically set the TCP buffer/window size.*
- union [globus\\_ftp\\_control\\_tcpbuffer\\_t](#)  
*control tcpbuffer attribute structure*
- struct [globus\\_ftp\\_control\\_auth\\_info\\_s](#)  
*Authentication Values.*

## Defines

- #define [GLOBUS\\_FTP\\_CONTROL\\_MODULE](#) (&globus\_i\_ftp\_control\_module)

## Typedefs

- typedef enum [globus\\_ftp\\_control\\_type\\_e](#) [globus\\_ftp\\_control\\_type\\_t](#)
- typedef enum [globus\\_ftp\\_control\\_mode\\_e](#) [globus\\_ftp\\_control\\_mode\\_t](#)
- typedef enum [globus\\_ftp\\_control\\_dcau\\_mode\\_e](#) [globus\\_ftp\\_control\\_dcau\\_mode\\_t](#)
- typedef [globus\\_ftp\\_control\\_dcau\\_subject\\_s](#) [globus\\_ftp\\_control\\_dcau\\_subject\\_t](#)
- typedef enum [globus\\_ftp\\_control\\_striping\\_mode\\_e](#) [globus\\_ftp\\_control\\_striping\\_mode\\_t](#)
- typedef [globus\\_ftp\\_control\\_round\\_robin\\_s](#) [globus\\_ftp\\_control\\_round\\_robin\\_t](#)
- typedef [globus\\_ftp\\_control\\_dcau\\_u](#) [globus\\_ftp\\_control\\_dcau\\_t](#)
- typedef [globus\\_ftp\\_control\\_layout\\_u](#) [globus\\_ftp\\_control\\_layout\\_t](#)
- typedef enum [globus\\_ftp\\_control\\_structure\\_e](#) [globus\\_ftp\\_control\\_structure\\_t](#)
- typedef enum [globus\\_ftp\\_control\\_parallelism\\_mode\\_e](#) [globus\\_ftp\\_control\\_parallelism\\_mode\\_t](#)
- typedef [globus\\_ftp\\_control\\_parallelism\\_u](#) [globus\\_ftp\\_control\\_parallelism\\_t](#)
- typedef enum [globus\\_ftp\\_control\\_tcpbuffer\\_mode\\_e](#) [globus\\_ftp\\_control\\_tcpbuffer\\_mode\\_t](#)
- typedef [globus\\_ftp\\_control\\_tcpbuffer\\_automatic\\_s](#) [globus\\_ftp\\_control\\_tcpbuffer\\_automatic\\_t](#)
- typedef [globus\\_ftp\\_control\\_tcpbuffer\\_t](#) [globus\\_ftp\\_control\\_tcpbuffer\\_t](#)
- typedef [globus\\_ftp\\_control\\_auth\\_info\\_s](#) [globus\\_ftp\\_control\\_auth\\_info\\_t](#)
- typedef void(\* [globus\\_ftp\\_control\\_response\\_callback\\_t](#) )(void \*callback\_arg, struct globus\_ftp\_control\_handle\_s \*handle, globus\_object\_t \*error, globus\_ftp\_control\_response\_t \*ftp\_response)
- typedef void(\* [globus\\_ftp\\_control\\_callback\\_t](#) )(void \*callback\_arg, struct globus\_ftp\_control\_handle\_s \*handle, globus\_object\_t \*error)
- typedef void(\* [globus\\_ftp\\_control\\_command\\_callback\\_t](#) )(void \*callback\_arg, struct globus\_ftp\_control\_handle\_s \*handle, globus\_object\_t \*error, union globus\_ftp\_control\_command\_u \*command)
- typedef void(\* [globus\\_ftp\\_control\\_auth\\_callback\\_t](#) )(void \*callback\_arg, struct globus\_ftp\_control\_handle\_s \*handle, globus\_object\_t \*error, [globus\\_ftp\\_control\\_auth\\_info\\_t](#) \*auth\_result)
- typedef unsigned long [globus\\_ftp\\_control\\_auth\\_requirements\\_t](#)

- typedef void(\* [globus\\_ftp\\_control\\_data\\_callback\\_t](#) )(void \*callback\_arg, globus\_ftp\_control\_handle\_t \*handle, globus\_object\_t \*error, globus\_byte\_t \*buffer, globus\_size\_t length, globus\_off\_t offset, globus\_bool\_t eof)
- typedef void(\* [globus\\_ftp\\_control\\_server\\_callback\\_t](#) )(void \*callback\_arg, struct globus\_ftp\_control\_server\_s \*server\_handle, globus\_object\_t \*error)

## Enumerations

- enum [globus\\_ftp\\_control\\_type\\_e](#)
- enum [globus\\_ftp\\_control\\_mode\\_e](#)
- enum [globus\\_ftp\\_control\\_dcau\\_mode\\_e](#)
- enum [globus\\_ftp\\_control\\_stripping\\_mode\\_e](#)
- enum [globus\\_ftp\\_control\\_protection\\_t](#)
- enum [globus\\_ftp\\_control\\_delay\\_passive\\_t](#)
- enum [globus\\_ftp\\_control\\_structure\\_e](#)
- enum [globus\\_ftp\\_control\\_parallelism\\_mode\\_e](#)
- enum [globus\\_ftp\\_control\\_tcpbuffer\\_mode\\_e](#) {  
[GLOBUS\\_FTP\\_CONTROL\\_TCPBUFFER\\_DEFAULT](#),  
[GLOBUS\\_FTP\\_CONTROL\\_TCPBUFFER\\_FIXED](#),  
[GLOBUS\\_FTP\\_CONTROL\\_TCPBUFFER\\_AUTOMATIC](#) }

## Functions

- globus\_result\_t [globus\\_ftp\\_control\\_local\\_layout](#) (globus\_ftp\_control\_handle\_t \*handle, [globus\\_ftp\\_control\\_layout\\_t](#) \*layout, globus\_size\_t data\_size)
- globus\_result\_t [globus\\_ftp\\_control\\_data\\_set\\_interface](#) (globus\_ftp\_control\_handle\_t \*handle, const char \*interface\_addr)
- globus\_result\_t [globus\\_ftp\\_control\\_create\\_data\\_info](#) (globus\_ftp\_control\_handle\_t \*handle, globus\_ftp\_control\_data\_write\_info\_t \*data\_info, globus\_byte\_t \*buffer, globus\_size\_t length, globus\_off\_t offset, globus\_bool\_t eof, [globus\\_ftp\\_control\\_data\\_callback\\_t](#) callback, void \*callback\_arg)
- globus\_result\_t [globus\\_ftp\\_control\\_release\\_data\\_info](#) (globus\_ftp\_control\_handle\_t \*handle, globus\_ftp\_control\_data\_write\_info\_t \*data\_info)
- globus\_result\_t [globus\\_ftp\\_control\\_data\\_write\\_stripe](#) (globus\_ftp\_control\_handle\_t \*handle, globus\_byte\_t \*buffer, globus\_size\_t length, globus\_off\_t offset, globus\_bool\_t eof, int stripe\_ndx, [globus\\_ftp\\_control\\_data\\_callback\\_t](#) callback, void \*callback\_arg)
- globus\_result\_t [globus\\_X\\_ftp\\_control\\_data\\_write\\_stripe](#) (globus\_ftp\_control\_handle\_t \*handle, globus\_byte\_t \*buffer, globus\_size\_t length, globus\_off\_t offset, globus\_bool\_t eof, int stripe\_ndx, globus\_ftp\_control\_data\_write\_info\_t \*data\_info)

### 6.2.1 Detailed Description

GSIFTP Control Connection API (Data structures and types).

### 6.2.2 Define Documentation

#### 6.2.2.1 #define GLOBUS\_FTP\_CONTROL\_MODULE (&globus\_i\_ftp\_control\_module)

Module descriptor.

The Globus FTP Control library uses the standard module activation and deactivation API to initialize it's state. Before any GSIFTP functions are called, the module must be activated

```
globus_module_activate(GLOBUS_GSIFTP_CONTROL_MODULE);
```

This function returns GLOBUS\_SUCCESS if the GSIFTP library was successfully initialized. This may be called multiple times.

To deactivate the GSIFTP library, the following must be called

```
globus_module_deactivate(GLOBUS_GSIFTP_CONTROL_MODULE);
```

### 6.2.3 Typedef Documentation

#### 6.2.3.1 typedef enum [globus\\_ftp\\_control\\_type\\_e](#) [globus\\_ftp\\_control\\_type\\_t](#)

control structure types.

The enumeration values match the character value of the argument to TYPE.

#### 6.2.3.2 typedef enum [globus\\_ftp\\_control\\_mode\\_e](#) [globus\\_ftp\\_control\\_mode\\_t](#)

control structure mode

#### 6.2.3.3 typedef enum [globus\\_ftp\\_control\\_dcau\\_mode\\_e](#) [globus\\_ftp\\_control\\_dcau\\_mode\\_t](#)

control dcau types

#### 6.2.3.4 typedef struct [globus\\_ftp\\_control\\_dcau\\_subject\\_s](#) [globus\\_ftp\\_control\\_dcau\\_subject\\_t](#)

control dcau subject authentication type

#### 6.2.3.5 typedef enum [globus\\_ftp\\_control\\_striping\\_mode\\_e](#) [globus\\_ftp\\_control\\_striping\\_mode\\_t](#)

control striping Types

#### 6.2.3.6 typedef struct [globus\\_ftp\\_control\\_round\\_robin\\_s](#) [globus\\_ftp\\_control\\_round\\_robin\\_t](#)

control striping round robin attribute structure

#### 6.2.3.7 typedef union [globus\\_ftp\\_control\\_dcau\\_u](#) [globus\\_ftp\\_control\\_dcau\\_t](#)

control dcau union

#### 6.2.3.8 typedef union [globus\\_ftp\\_control\\_layout\\_u](#) [globus\\_ftp\\_control\\_layout\\_t](#)

control striping attribute union

#### 6.2.3.9 typedef enum [globus\\_ftp\\_control\\_structure\\_e](#) [globus\\_ftp\\_control\\_structure\\_t](#)

control structure structure

#### 6.2.3.10 typedef enum [globus\\_ftp\\_control\\_parallelism\\_mode\\_e](#) [globus\\_ftp\\_control\\_parallelism\\_mode\\_t](#)

control parallelism Types



#### 6.2.3.11 **typedef union** [globus\\_ftp\\_control\\_parallelism\\_u](#) [globus\\_ftp\\_control\\_parallelism\\_t](#)

control parallelism attribute structure

#### 6.2.3.12 **typedef enum** [globus\\_ftp\\_control\\_tcpbuffer\\_mode\\_e](#) [globus\\_ftp\\_control\\_tcpbuffer\\_mode\\_t](#)

TCP Buffer Setting Modes.

#### 6.2.3.13 **typedef struct** [globus\\_ftp\\_control\\_tcpbuffer\\_automatic\\_s](#) [globus\\_ftp\\_control\\_tcpbuffer\\_automatic\\_t](#)

Automatically set the TCP buffer/window size.

#### 6.2.3.14 **typedef union** [globus\\_ftp\\_control\\_tcpbuffer\\_t](#) [globus\\_ftp\\_control\\_tcpbuffer\\_t](#)

control tcpbuffer attribute structure

#### 6.2.3.15 **typedef struct** [globus\\_ftp\\_control\\_auth\\_info\\_s](#) [globus\\_ftp\\_control\\_auth\\_info\\_t](#)

Authentication Values.

This structure is populated and passed back to the user via the [globus\\_ftp\\_control\\_auth\\_callback\\_t\(\)](#). It contains the information needed to decide if a client may use the server.

#### 6.2.3.16 **typedef void**(\* [globus\\_ftp\\_control\\_response\\_callback\\_t](#))( **void** \* [callback\\_arg](#), **struct** [globus\\_ftp\\_control\\_handle\\_s](#) \* [handle](#), [globus\\_object\\_t](#) \* [error](#), [globus\\_ftp\\_control\\_response\\_t](#) \* [ftp\\_response](#))

Asynchronous operation completion callback.

This callback is called whenever a reply to command is received on the FTP control channel. It allows the user to handle the received reply or alternatively handle any errors that occurred during the interaction with the FTP server. This function will be called multiple times in the case when intermediate responses (1yz) are received.

##### **Parameters:**

***callback\_arg*** User supplied argument to the callback function

***handle*** A pointer to the GSIFTP control handle. Used to identify which control connection the operation was applied to.

***error*** Pointer to a globus error object containing information about any errors that occurred processing the operation

***ftp\_response*** Pointer to a response structure containing the FTP response to the command.

#### 6.2.3.17 **typedef void**(\* [globus\\_ftp\\_control\\_callback\\_t](#))( **void** \* [callback\\_arg](#), **struct** [globus\\_ftp\\_control\\_handle\\_s](#) \* [handle](#), [globus\\_object\\_t](#) \* [error](#))

Asynchronous control callback.

This callback is used as a generic control operation callback.

##### **Parameters:**

***callback\_arg*** User supplied argument to the callback function

***handle*** A pointer to the GSIFTP control handle. Used to identify which control connection the operation was applied to.

***error*** Pointer to a globus error object containing information about any errors that occurred processing the operation

**6.2.3.18** typedef void(\* [globus\\_ftp\\_control\\_command\\_callback\\_t](#))( void \* callback\_arg, struct globus\_ftp\_control\_handle\_s \* handle, globus\_object\_t \* error, union globus\_ftp\_control\_command\_u \* command)

Server command callback.

When a command from a client is received on the control channel a user callback with this signature is called.

**Parameters:**

*callback\_arg* The user argument passed to the callback function.

*handle* The control handle that the command was issued on.

*error* Indicates if a command was successful read or or if a failure occurred. This object will be freed once this callback returns. If the user wishes to have a copy of the error that persists past the life of this callback, they must make a copy using [globus\\_object\\_copy\(\)](#), and free it with [globus\\_object\\_free\(\)](#).

*command* The command structure indicates what type of command the client issued. Based on the 'type' further information can be extracted. This command structure will be freed once this callback returns. If the user wishes to have a copy of the error that persists past the life of this callback, they must make a copy using [globus\\_ftp\\_control\\_command\\_copy\(\)](#), and free it with [globus\\_ftp\\_control\\_command\\_free\(\)](#).

**6.2.3.19** typedef void(\* [globus\\_ftp\\_control\\_auth\\_callback\\_t](#))( void \* callback\_arg, struct globus\_ftp\_control\_handle\_s \* handle, globus\_object\_t \* error, [globus\\_ftp\\_control\\_auth\\_info\\_t](#) \* auth\_result)

Server authentication complete callback.

A function with this signature is registered by calling [globus\\_ftp\\_control\\_accept\(\)](#). It is called when the authentication protocol has completed. Based on the [auth\\_result](#), the server implementor should determine authorization and then send the appropriate response using [globus\\_ftp\\_control\\_send\\_response\(\)](#), indicating to the client whether authorization was successful or not.

**Parameters:**

*handle* This structure is populated when the callback is called and represents a control connection to the client.

*auth\_result* A [globus\\_ftp\\_control\\_auth\\_result\\_t](#) containing the values the client sent for gss authentication, user name, password and account. If any of the values were not sent by the client they will be NULL. Based on that information the user can decide if the client will be authorized for use of the server.

*callback\_arg* The user argument passed to the callback.

**6.2.3.20** typedef unsigned long [globus\\_ftp\\_control\\_auth\\_requirements\\_t](#)

Authentication requirements.

The value of this should be a bitwise or of

- GLOBUS\_FTP\_CONTROL\_AUTH\_NONE
- GLOBUS\_FTP\_CONTROL\_AUTH\_GSSAPI
- GLOBUS\_FTP\_CONTROL\_AUTH\_USER
- GLOBUS\_FTP\_CONTROL\_AUTH\_PASS
- GLOBUS\_FTP\_CONTROL\_AUTH\_ACCT

**6.2.3.21** `typedef void(* globus_ftp_control_data_callback_t)( void * callback_arg, globus_ftp_control_handle_t * handle, globus_object_t * error, globus_byte_t * buffer, globus_size_t length, globus_off_t offset, globus_bool_t eof)`

Asynchronous data transmission operation callback.

This callback is called in functions that send or receive data on the data channel(s).

In the case of a write, this function is invoked when the entire data buffer is sent. Depending on the data transfer properties set by the `globus_ftp_control_local_*`(`*`) functions, the data may actually be split into multiple buffers and sent to multiple data nodes.

In the case of a read, this function will return a single extent of the data. The order of the data returned is not defined in an extended block mode data transfer. It is up to the user of the API to re-construct the file order.

**Parameters:**

*callback\_arg* User supplied argument to the callback function

*handle* A pointer to the GSIFTP control handle. Used to identify which control connection the operation was applied to.

*error* Pointer to a globus error object containing information about any errors that occurred processing the operation

*buffer* The user buffer passed as a parameter to `globus_ftp_control_data_read()` or `globus_ftp_control_data_write()`.

*length* The amount of data in the buffer. In the case of an incoming data channel, this may be less than the buffer size.

*offset* The file offset of the data which is contained in the buffer.

*eof* This is set to `GLOBUS_TRUE` then all of the data associated with the transfer has arrived on the data connections associated with this handle. If multiple data callbacks are registered with this handle, there is no guaranteed order of the EOF callback with respect to other data callbacks. If multiple callbacks are registered when EOF is reached on the data connections, at least one callback function will be called with eof set to `GLOBUS_TRUE`.

**6.2.3.22** `typedef void(* globus_ftp_control_server_callback_t)( void * callback_arg, struct globus_ftp_control_server_s * server_handle, globus_object_t * error)`

Server callback.

A functions with this signature can be used as general callbacks for the GSIFTP server API.

**Parameters:**

*server\_handle* The server handle associated with callback.

*result* Indicates if the operation completed successfully or if a failure occurred.

*callback\_arg* The user argument passed to the callback function.

## 6.2.4 Enumeration Type Documentation

**6.2.4.1** `enum globus_ftp_control_type_e`

control structure types.

The enumeration values match the character value of the argument to `TYPE`.

**6.2.4.2** `enum globus_ftp_control_mode_e`

control structure mode

#### 6.2.4.3 enum [globus\\_ftp\\_control\\_dcau\\_mode\\_e](#)

control dcau types

#### 6.2.4.4 enum [globus\\_ftp\\_control\\_striping\\_mode\\_e](#)

control striping Types

#### 6.2.4.5 enum [globus\\_ftp\\_control\\_protection\\_t](#)

control protection levels

#### 6.2.4.6 enum [globus\\_ftp\\_control\\_delay\\_passive\\_t](#)

delayed passive flags

#### 6.2.4.7 enum [globus\\_ftp\\_control\\_structure\\_e](#)

control structure structure

#### 6.2.4.8 enum [globus\\_ftp\\_control\\_parallelism\\_mode\\_e](#)

control parallelism Types

#### 6.2.4.9 enum [globus\\_ftp\\_control\\_tcpbuffer\\_mode\\_e](#)

TCP Buffer Setting Modes.

#### Enumeration values:

***GLOBUS\_FTP\_CONTROL\_TCPBUFFER\_DEFAULT*** Don't change the TCP buffer/window size from the system default.

***GLOBUS\_FTP\_CONTROL\_TCPBUFFER\_FIXED*** Set the TCP buffer/window size to a fixed value.

***GLOBUS\_FTP\_CONTROL\_TCPBUFFER\_AUTOMATIC*** Automatically set the TCP buffer/window size.

### 6.2.5 Function Documentation

#### 6.2.5.1 [globus\\_result\\_t](#) [globus\\_ftp\\_control\\_local\\_layout](#) ([globus\\_ftp\\_control\\_handle\\_t](#) \* *handle*, [globus\\_ftp\\_control\\_layout\\_t](#) \* *layout*, [globus\\_size\\_t](#) *data\_size*)

Update the handle with the layout and the size of the data sent over the data channel.

This function is deprecated. The interface will be changed to that of [globus\\_X\\_ftp\\_control\\_local\\_layout\(\)](#)

#### Parameters:

***handle*** A pointer to the FTP control handle into which to insert the layout information.

***layout*** A variable containing the layout information

***data\_size*** The size of the data that is going to be sent. This may be needed to interpret the layout information.

### 6.2.5.2 `globus_result_t globus_ftp_control_data_set_interface (globus_ftp_control_handle_t * handle, const char * interface_addr)`

Create an outgoing FTP data connection.

This function sets the interface that will be used to send and receive information along the data channel.

#### Parameters:

*handle* A pointer to a FTP control handle which is configured to create an outgoing data connection.

*interface\_addr*

### 6.2.5.3 `globus_result_t globus_ftp_control_create_data_info (globus_ftp_control_handle_t * handle, globus_ftp_control_data_write_info_t * data_info, globus_byte_t * buffer, globus_size_t length, globus_off_t offset, globus_bool_t eof, globus\_ftp\_control\_data\_callback\_t callback, void * callback_arg)`

Create a `globus_ftp_control_data_write_info_t` structure.

This function populates a `globus_ftp_control_data_callback_t` structure with valid information. This structure provides the user a way to register several data writes with a single callback. This is quite useful to the writer of enqueue functions. It allows a single call to [globus\\_ftp\\_control\\_data\\_write\(\)](#) to be broken up into many writes, potentially on different stripes, and for a single callback to be called when all are finished.

#### Parameters:

*handle* A pointer to a FTP control handle. The handle contains information about the current state of the control and data connections.

*data\_info* The `globus_ftp_control_data_write_info_t` structure to be released.

*buffer* The pointer to the user buffer that will be passed to the callback argument when there are zero references to *data\_info*. This is intended to be the start of all the data the user intends to write using [globus\\_ftp\\_control\\_data\\_write\\_stripe\(\)](#), but it does not have to be.

*length* The length of the memory segment pointed to by the argument buffer.

*offset* The file offset of the data segment specified.

*eof* This should be set to true if the user plans on registering eof on the *data\_info* structure.

*callback* The user function to be called when all references to *data\_info* are released. This occurs after all data registered for write from `globus_ftp_control_data_write_stripe` have occurred and the user calls [globus\\_ftp\\_control\\_release\\_data\\_info\(\)](#). The callback is passed all of the arguments passed to this function with the exception of *data\_info*.

*callback\_arg* User supplied argument to the callback function

### 6.2.5.4 `globus_result_t globus_ftp_control_release_data_info (globus_ftp_control_handle_t * handle, globus_ftp_control_data_write_info_t * data_info)`

Release a *data\_info* structure.

This function releases all memory and references created when a call to [globus\\_ftp\\_control\\_create\\_data\\_info\(\)](#) was made. For every call to [globus\\_ftp\\_control\\_create\\_data\\_info\(\)](#) a call to this function must be made.

#### Parameters:

*handle* A pointer to a FTP control handle. The handle contains information about the current state of the control and data connections.

*data\_info* The `globus_ftp_control_data_write_info_t` structure to be released.

**6.2.5.5 globus\_result\_t globus\_ftp\_control\_data\_write\_stripe** (globus\_ftp\_control\_handle\_t \* *handle*, globus\_byte\_t \* *buffer*, globus\_size\_t *length*, globus\_off\_t *offset*, globus\_bool\_t *eof*, int *stripe\_ndx*, globus\_ftp\_control\_data\_callback\_t *callback*, void \* *callback\_arg*)

Write FTP data to a particular stripe.

This function allows the user to write to a specified stripe. The stripe index relates to the order passed into local\_spor(). This function differs from globus\_ftp\_control\_data\_write() in that no enqueue function is needed since the user specifies the stripe on which data is written. In order to use this function the user must have a valid pointer to a globus\_ftp\_control\_data\_write\_info\_t structure. The data\_info structure can be obtained by a call to globus\_ftp\_control\_create\_data\_info(). Many calls to this function can be made, but only a single user callback occurs per creation of a globus\_ftp\_control\_data\_write\_info\_t structure.

**Parameters:**

*handle* A pointer to a FTP control handle. The handle contains information about the current state of the control and data connections.

*buffer* a pointer to the data the user wishes to send along the FTP data channels.

*length* the length of the data pointer to by the parameter buffer.

*offset* the offset into the file of the data.

*eof* A boolean stating that this will be the last chunk of data registered on the given stripe. In order to properly send an eof message the user must register an eof on every stripe.

*stripe\_ndx* The index of the stripe on which the data will be sent. The index of each stripe is determined by the call to local\_spas or local\_spor.

*callback* The function to be called once the data has been sent

*callback\_arg* User supplied argument to the callback function

**6.2.5.6 globus\_result\_t globus\_X\_ftp\_control\_data\_write\_stripe** (globus\_ftp\_control\_handle\_t \* *handle*, globus\_byte\_t \* *buffer*, globus\_size\_t *length*, globus\_off\_t *offset*, globus\_bool\_t *eof*, int *stripe\_ndx*, globus\_ftp\_control\_data\_write\_info\_t \* *data\_info*)

Write data on a specific stripe from an enqueue callback function only.

This function allows the user to register the write of ftp data on a specific stripe. This function can only be called from an enqueue function callback. This function should be used only by the implementor of an enqueue function. It should be viewed as unstable and used only by advanced users. This is the only function in the library that the enqueue function implementor is allowed from the enqueue callback.

**Parameters:**

*handle* A pointer to a FTP control handle. The handle contains information about the current state of the control and data connections.

*buffer* a pointer to the data the user wishes to send along the FTP data channels.

*length* the length of the data pointer to by the parameter buffer.

*offset* the offset into the file of the data.

*eof* a boolean stating that this is the last buffer to be registered. When using the \_X\_ version of this function the user does not need to register an eof on each stripe, the control library will take care of that internally.

*stripe\_ndx* The index of the stripe on which the data will be sent. The index of each stripe is determined by the call to local\_spas or local\_spor.

*data\_info* An opaque structure that is passed into the enqueue function and contains reference count and state information. The same data\_info pointer that is passed into the enqueue function must be used for this parameter.

## 6.3 globus\_ftp\_control\_client.c File Reference

Client-side FTP Control API.

### Functions

- globus\_result\_t [globus\\_ftp\\_control\\_handle\\_init](#) (globus\_ftp\_control\_handle\_t \*handle)
- globus\_result\_t [globus\\_ftp\\_control\\_handle\\_destroy](#) (globus\_ftp\_control\_handle\_t \*handle)
- globus\_result\_t [globus\\_ftp\\_control\\_connect](#) (globus\_ftp\_control\_handle\_t \*handle, char \*host, unsigned short port, [globus\\_ftp\\_control\\_response\\_callback\\_t](#) callback, void \*callback\_arg)
- globus\_result\_t [globus\\_ftp\\_control\\_response\\_destroy](#) (globus\_ftp\_control\_response\_t \*response)
- globus\_result\_t [globus\\_ftp\\_control\\_response\\_copy](#) (globus\_ftp\_control\_response\_t \*src, globus\_ftp\_control\_response\_t \*dest)
- globus\_result\_t [globus\\_ftp\\_control\\_authenticate](#) (globus\_ftp\_control\_handle\_t \*handle, [globus\\_ftp\\_control\\_auth\\_info\\_t](#) \*auth\_info, globus\_bool\_t use\_auth, [globus\\_ftp\\_control\\_response\\_callback\\_t](#) callback, void \*callback\_arg)
- globus\_result\_t [globus\\_ftp\\_control\\_send\\_command](#) (globus\_ftp\_control\_handle\_t \*handle, const char \*cmdspec, [globus\\_ftp\\_control\\_response\\_callback\\_t](#) callback, void \*callback\_arg,...)
- globus\_result\_t [globus\\_ftp\\_control\\_abort](#) (globus\_ftp\_control\_handle\_t \*handle, [globus\\_ftp\\_control\\_response\\_callback\\_t](#) callback, void \*callback\_arg)
- globus\_result\_t [globus\\_ftp\\_control\\_quit](#) (globus\_ftp\_control\_handle\_t \*handle, [globus\\_ftp\\_control\\_response\\_callback\\_t](#) callback, void \*callback\_arg)
- globus\_result\_t [globus\\_ftp\\_control\\_force\\_close](#) (globus\_ftp\_control\_handle\_t \*handle, [globus\\_ftp\\_control\\_response\\_callback\\_t](#) callback, void \*callback\_arg)
- globus\_result\_t [globus\\_ftp\\_control\\_auth\\_info\\_init](#) (globus\_ftp\_control\_auth\_info\_t \*auth\_info, gss\_cred\_id\_t credential\_handle, globus\_bool\_t encrypt, char \*user, char \*password, char \*account, char \*subject)
- int [globus\\_ftp\\_control\\_auth\\_info\\_compare](#) (globus\_ftp\_control\_auth\_info\_t \*auth\_info\_1, [globus\\_ftp\\_control\\_auth\\_info\\_t](#) \*auth\_info\_2)

### 6.3.1 Detailed Description

Client-side FTP Control API.

### 6.3.2 Function Documentation

#### 6.3.2.1 globus\_result\_t globus\_ftp\_control\_handle\_init (globus\_ftp\_control\_handle\_t \* *handle*)

Initialize a globus ftp handle.

This function will set up (i.e. initialize all mutexes and variables) a globus ftp handle. It will also enter the handle in a list used by the module activation/deactivation functions.

#### Parameters:

*handle* The handle to initialize.

#### Returns:

- GLOBUS\_SUCCESS
- error object

### 6.3.2.2 `globus_result_t globus_ftp_control_handle_destroy (globus_ftp_control_handle_t * handle)`

Destroy a globus ftp handle.

This function will free up all dynamically allocated memory associated with a given globus ftp handle. It will also remove the handle from a list used by the module activation/deactivation functions. This function should only be called after a call to either `globus_ftp_control_force_close` or `globus_ftp_control_quit`.

#### Parameters:

*handle* The handle to destroy.

#### Returns:

- success
- invalid handle
- handle is still in connected state

### 6.3.2.3 `globus_result_t globus_ftp_control_connect (globus_ftp_control_handle_t * handle, char * host, unsigned short port, globus\_ftp\_control\_response\_callback\_t callback, void * callback_arg)`

Create a new control connection to an FTP server.

This function is used to initiate an FTP control connection. It creates the socket to the FTP server. When the connection is made to the server, and the server's identification string is received, the callback function will be invoked.

#### Parameters:

*handle* A pointer to a initialized FTP control handle. This handle will be used for all subsequent FTP control operations.

*host* The hostname of the FTP server.

*port* The TCP port number of the FTP server.

*callback* A function to be called once the connection to the server is established, and a response has been read.

*callback\_arg* Parameter to the callback function.

#### Returns:

- success
- Null handle
- Null host
- Illegal port number
- Null callback
- Cannot resolve hostname
- Cannot create socket

#### Callback errors:

- success
- connection refused
- protocol error
- eof

#### Expected callback response values:

- 120 Service ready in nnn minutes.



- 220 Service ready for new user.
- 421 Service not available, closing control connection.
- 500 Syntax error, command unrecognized.

**Note:**

The server may send other responses.

#### 6.3.2.4 `globus_result_t globus_ftp_control_response_destroy (globus_ftp_control_response_t * response)`

Helper function which frees the memory associated with a response structure.

This is a helper function which frees the memory associated with a response structure.

**Parameters:**

*response* This parameter indicates the response structure to destroy

**Returns:**

- Error object
- GLOBUS\_SUCCESS

#### 6.3.2.5 `globus_result_t globus_ftp_control_response_copy (globus_ftp_control_response_t * src, globus_ftp_control_response_t * dest)`

Helper function which copies one response structure to another.

This is a helper function which copies one response structure to another.

**Parameters:**

*src* This parameter indicates the response structure to copy

*dest* This parameter specifies the target response structure

**Returns:**

- Error object
- GLOBUS\_SUCCESS

#### 6.3.2.6 `globus_result_t globus_ftp_control_authenticate (globus_ftp_control_handle_t * handle, globus_ftp_control_auth_info_t * auth_info, globus_bool_t use_auth, globus_ftp_control_response_callback_t callback, void * callback_arg)`

Authenticate the user to the FTP server.

This will perform the authentication handshake with the FTP server. depending on which parameters are non-NULL, the authentication may involve GSSAPI credentials, a username, a password, and an account name.

**Note:**

Do we want to add attribute arguments for:

- specifying type of delegation
- gsswrap control messages for integrity or confidentiality

**Parameters:**

*handle* A pointer to a unauthenticated GSIFTP control handle. In the case of GSS authentication the GSS security context is stored in this structure.

***auth\_info*** This structure is used to pass the following information:

- **user** The user's name for login purposes. If this string is "anonymous", "ftp", GLOBUS\_NULL or ":globus-mapping:" then the password argument is optional. If this string is GLOBUS\_NULL or ":globus-mapping:" and gss\_auth is true then the users login is looked by the FTP server host.
- **password** The password for the above user argument. If the user argument is "anonymous" or "ftp" or if gss\_auth is true this string may be GLOBUS\_NULL.
- **account** This parameter is optional. If not used it should be set to GLOBUS\_NULL. It might be needed by firewalls.
- **auth\_gssapi\_subject** The GSSAPI subject name of the server you are connecting to. If this is GLOBUS\_NULL, and the gss\_auth parameter is set to GLOBUS\_TRUE, then the name will default to the host name.

***use\_auth*** If set to GLOBUS\_TRUE the above argument indicates that GSS authentication should be used, otherwise cleartext user/password authentication is used.

***callback*** The function to be called once the authentication process is complete or when an error occurs.

***callback\_arg*** User supplied argument to the callback function

**Returns:**

- success
- Null handle
- Invalid handle
- Handle already authenticated

**Callback errors:**

- success
- authentication failed
- protocol error
- eof

**Expected callback response values:**

- 230 User logged in, proceed.
- 232 User logged in, authorized by security data exchange.
- 234 Security data exchange complete.
- 331 User name okay, need password.
- 332 Need account for login.
- 336 Username okay, need password. Challenge is "...."
- 431 Need some unavailable resource to process security.
- 500 Syntax error, command unrecognized.
- 530 Not logged in.

**Note:**

The server may send other responses.

**6.3.2.7** `globus_result_t globus_ftp_control_send_command (globus_ftp_control_handle_t * handle, const char * cmdspec, globus\_ftp\_control\_response\_callback\_t callback, void * callback_arg, ...)`

Send an FTP protocol command to the FTP server and register a response handler.

This function is used to send an FTP command, and register a handler to receive the FTP reply (or replies, if an intermediate one is sent). When the control channel is gss authenticated, the message and the reply will be automatically gss wrapped/unwrapped.

**Parameters:**

- handle*** A pointer to a GSIFTP control handle. The command described by the cmdspec is issued to the server over the control channel associated with this handle.
- cmdspec*** A printf-style format string containing the text of the command to send to the server. The optional parameters to the format string are passed after the callback\_arg in the function invocation.
- callback*** The function to be called once the authentication process is complete or when an error occurs.
- callback\_arg*** User supplied argument to the callback function
- ... Parameters which will be substituted into the escapes in the cmdspec string.

**Returns:**

- Success
- Null handle
- Command already in progress

**Callback errors:**

- success
- protocol error
- eof

**Expected callback response values:**

Any defined in RFC 959, 2228, 2389, draft-ietf-ftpext-mlst-10, or the [protocol extensions](#) document.

**6.3.2.8 globus\_result\_t globus\_ftp\_control\_abort (globus\_ftp\_control\_handle\_t \* handle, [globus\\_ftp\\_control\\_response\\_callback\\_t](#) callback, void \* callback\_arg)**

Send an ABORT to the FTP server and register a response handler.

This function is used to send the ABORT message to the FTP server. The ABORT message is sent out-of-band, and terminates any current data transfer in progress.

As a result of the ABORT, the data channels used by this control channel will be closed. The data command callback will be issued with either a completion reply, or a transfer aborted reply. The ABORT callback will also be invoked, with the server's response to the abort command.

Any attempts to register buffers for read or write after an ABORT has been sent will fail with a "no transfer in progress" error.

**Parameters:**

- handle*** A pointer to a GSIFTP control handle. The ABORT command is issued to the server over the control channel associated with this handle.
- callback*** The function to be called once the authentication process is complete or when an error occurs.
- callback\_arg*** User supplied argument to the callback function

**Returns:**

- Success
- Null handle
- No transfer in progress

**Callback errors:**

- success
- protocol error
- eof

**Expected callback response values:**

- 226 Abort successful.
- 500 Syntax error, command unrecognized.

**Note:**

The server may send other responses.

**6.3.2.9 globus\_result\_t globus\_ftp\_control\_quit (globus\_ftp\_control\_handle\_t \* *handle*, globus\_ftp\_control\_response\_callback\_t *callback*, void \* *callback\_arg*)**

Send a QUIT message to the FTP server and register a response handler.

This function is used to close the control channel to the FTP server. There should be no transfer commands in progress when this is called. Once the final response callback passed to this function is invoked, the control handle can no longer be used for any gsiftp control operations.

**Note:**

Need to further define behavior for when a QUIT happens during a transfer or command is in progress. Since this function waits until all other callbacks are completed before calling it's own callback it may not be called in a blocking fashion from another callback.

**Parameters:**

*handle* A pointer to a GSIFTP control handle. The quit message is issued to the server over the control channel associated with this handle.

*callback* The function to be called once the authentication process is complete or when an error occurs.

*callback\_arg* User supplied argument to the callback function

**Returns:**

- Success
- Null handle
- Command in progress

**Callback errors:**

- success
- protocol error
- eof

**Expected callback response values:**

- 221 Service closing control connection.
- 500 Syntax error, command unrecognized.

**Note:**

The server may send other responses.

**6.3.2.10 globus\_result\_t globus\_ftp\_control\_force\_close (globus\_ftp\_control\_handle\_t \* *handle*, globus\_ftp\_control\_response\_callback\_t *callback*, void \* *callback\_arg*)**

Force a close of the control connection without waiting for outstanding commands to complete and without sending QUIT.

This function is used to close the control channel to the FTP server. Once the final response callback passed to this function is invoked, the control handle can no longer be used for any gsiftp control operations.

**Note:**

Since this function waits until all other callbacks are completed before calling it's own callback it may not be called in a blocking fashion from another callback.

**Parameters:**

***handle*** A pointer to a GSIFTP control handle. The quit message is issued to the server over the control channel associated with this handle.

***callback*** The function to be called once the authentication process is complete or when an error occurs.

***callback\_arg*** User supplied argument to the callback function

**Returns:**

- Success
- Null handle

**Callback errors:**

- success
- failure

**Expected callback response values:**

- GLOBUS\_NULL

**6.3.2.11** `globus_result_t globus_ftp_control_auth_info_init (globus_ftp_control_auth_info_t * auth_info, gss_cred_id_t credential_handle, globus_bool_t encrypt, char * user, char * password, char * account, char * subject)`

Helper function which initializes a authentication information structure.

This is helper function initializes a authentication information structure with the values contained in the second to fifth arguments, which may be GLOBUS\_NULL. No memory is allocated in this function.

**Parameters:**

***auth\_info*** The authentication structure to initialize.

***credential\_handle*** The credential to use for authentication. This may be GSS\_C\_NO\_CREDENTIAL to use the user's default credential.

***encrypt*** Boolean whether or not to encrypt the control channel for this handle.

***user*** The user name

***password*** The password for the user name

***account*** The account for the user name/password

***subject*** The gss api subject name

**Returns:**

- Error object
- GLOBUS\_SUCCESS

**6.3.2.12** `int globus_ftp_control_auth_info_compare (globus_ftp_control_auth_info_t * auth_info_1, globus_ftp_control_auth_info_t * auth_info_2)`

Helper function which compares two authentication information structures.

This is helper function compares two authentication information structures and return zero if the two structures are deemed equal and a non-zero value otherwise.

**Parameters:**

- auth\_info\_1* The first authentication structure
- auth\_info\_2* The second authentication structure

**Returns:**

- 0 if the structures are equal
- !=0 if the structures differ or an error occurred

## 6.4 globus\_ftp\_control\_data.c File Reference

FTP Data Connection Configuration and Management.

**Functions**

- globus\_result\_t [globus\\_ftp\\_control\\_data\\_connect\\_read](#) (globus\_ftp\_control\_handle\_t \*handle, globus\_ftp\_control\_data\_connect\_callback\_t callback, void \*user\_arg)
- globus\_result\_t [globus\\_ftp\\_control\\_data\\_set\\_interface](#) (globus\_ftp\_control\_handle\_t \*handle, const char \*interface\_addr)
- globus\_result\_t [globus\\_ftp\\_control\\_data\\_connect\\_write](#) (globus\_ftp\_control\_handle\_t \*handle, globus\_ftp\_control\_data\_connect\_callback\_t callback, void \*user\_arg)
- globus\_result\_t [globus\\_ftp\\_control\\_data\\_add\\_channels](#) (globus\_ftp\_control\_handle\_t \*handle, unsigned int num\_channels, unsigned int stripe\_ndx)
- globus\_result\_t [globus\\_ftp\\_control\\_data\\_send\\_eof](#) (globus\_ftp\_control\_handle\_t \*handle, int count[], int array\_size, globus\_bool\_t eof\_message, [globus\\_ftp\\_control\\_callback\\_t](#) cb, void \*user\_arg)
- globus\_result\_t [globus\\_ftp\\_control\\_data\\_remove\\_channels](#) (globus\_ftp\_control\_handle\_t \*handle, unsigned int num\_channels, unsigned int stripe\_ndx)
- globus\_result\_t [globus\\_ftp\\_control\\_data\\_query\\_channels](#) (globus\_ftp\_control\_handle\_t \*handle, unsigned int \*num\_channels, unsigned int stripe\_ndx)
- globus\_result\_t [globus\\_ftp\\_control\\_data\\_get\\_total\\_data\\_channels](#) (globus\_ftp\_control\_handle\_t \*handle, unsigned int \*num\_channels, unsigned int stripe\_ndx)
- globus\_result\_t [globus\\_ftp\\_control\\_local\\_send\\_eof](#) (globus\_ftp\_control\_handle\_t \*handle, globus\_bool\_t send\_eof)
- globus\_result\_t [globus\\_ftp\\_control\\_local\\_parallelism](#) (globus\_ftp\_control\_handle\_t \*handle, [globus\\_ftp\\_control\\_parallelism\\_t](#) \*parallelism)
- globus\_result\_t [globus\\_ftp\\_control\\_local\\_pasv](#) (globus\_ftp\_control\_handle\_t \*handle, globus\_ftp\_control\_host\_port\_t \*address)
- globus\_result\_t [globus\\_ftp\\_control\\_local\\_spas](#) (globus\_ftp\_control\_handle\_t \*handle, globus\_ftp\_control\_host\_port\_t addresses[], unsigned int num\_addresses)
- globus\_result\_t [globus\\_ftp\\_control\\_local\\_port](#) (globus\_ftp\_control\_handle\_t \*handle, globus\_ftp\_control\_host\_port\_t \*address)
- globus\_result\_t [globus\\_ftp\\_control\\_local\\_spor](#) (globus\_ftp\_control\_handle\_t \*handle, globus\_ftp\_control\_host\_port\_t addresses[], unsigned int num\_addresses)
- globus\_result\_t [globus\\_ftp\\_control\\_local\\_type](#) (globus\_ftp\_control\_handle\_t \*handle, [globus\\_ftp\\_control\\_type\\_t](#) type, int form\_code)
- globus\_result\_t [globus\\_ftp\\_control\\_local\\_mode](#) (globus\_ftp\_control\_handle\_t \*handle, [globus\\_ftp\\_control\\_mode\\_t](#) mode)
- globus\_result\_t [globus\\_ftp\\_control\\_local\\_tcp\\_buffer](#) (globus\_ftp\_control\_handle\_t \*handle, [globus\\_ftp\\_control\\_tcpbuffer\\_t](#) \*tcp\_buffer)
- globus\_result\_t [globus\\_ftp\\_control\\_local\\_dcau](#) (globus\_ftp\_control\_handle\_t \*handle, const [globus\\_ftp\\_control\\_dcau\\_t](#) \*dcau, gss\_cred\_id\_t delegated\_credential\_handle)
- globus\_result\_t [globus\\_ftp\\_control\\_local\\_pbsz](#) (globus\_ftp\_control\_handle\_t \*handle, unsigned long buf\_size)

- globus\_result\_t [globus\\_ftp\\_control\\_get\\_pbsz](#) (globus\_ftp\_control\_handle\_t \*handle, unsigned long \*bufsize)
- globus\_result\_t [globus\\_ftp\\_control\\_local\\_stru](#) (globus\_ftp\_control\_handle\_t \*handle, [globus\\_ftp\\_control\\_structure\\_t](#) structure)
- globus\_result\_t [globus\\_ftp\\_control\\_data\\_write](#) (globus\_ftp\_control\_handle\_t \*handle, globus\_byte\_t \*buffer, globus\_size\_t length, globus\_off\_t offset, globus\_bool\_t eof, [globus\\_ftp\\_control\\_data\\_callback\\_t](#) callback, void \*callback\_arg)
- globus\_result\_t [globus\\_ftp\\_control\\_data\\_read](#) (globus\_ftp\_control\_handle\_t \*handle, globus\_byte\_t \*buffer, globus\_size\_t max\_length, [globus\\_ftp\\_control\\_data\\_callback\\_t](#) callback, void \*callback\_arg)
- globus\_result\_t [globus\\_ftp\\_control\\_local\\_layout](#) (globus\_ftp\_control\_handle\_t \*handle, [globus\\_ftp\\_control\\_layout\\_t](#) \*layout, globus\_size\_t data\_size)
- globus\_result\_t [globus\\_ftp\\_control\\_create\\_data\\_info](#) (globus\_ftp\_control\_handle\_t \*handle, globus\_ftp\_control\_data\_write\_info\_t \*data\_info, globus\_byte\_t \*buffer, globus\_size\_t length, globus\_off\_t offset, globus\_bool\_t eof, [globus\\_ftp\\_control\\_data\\_callback\\_t](#) callback, void \*callback\_arg)
- globus\_result\_t [globus\\_ftp\\_control\\_release\\_data\\_info](#) (globus\_ftp\_control\_handle\_t \*handle, globus\_ftp\_control\_data\_write\_info\_t \*data\_info)
- globus\_result\_t [globus\\_ftp\\_control\\_data\\_write\\_stripe](#) (globus\_ftp\_control\_handle\_t \*handle, globus\_byte\_t \*buffer, globus\_size\_t length, globus\_off\_t offset, globus\_bool\_t eof, int stripe\_ndx, [globus\\_ftp\\_control\\_data\\_callback\\_t](#) callback, void \*callback\_arg)
- globus\_result\_t [globus\\_X\\_ftp\\_control\\_data\\_write\\_stripe](#) (globus\_ftp\_control\_handle\_t \*handle, globus\_byte\_t \*buffer, globus\_size\_t length, globus\_off\_t offset, globus\_bool\_t eof, int stripe\_ndx, globus\_ftp\_control\_data\_write\_info\_t \*data\_info)
- globus\_result\_t [globus\\_ftp\\_control\\_data\\_force\\_close](#) (globus\_ftp\_control\_handle\_t \*control\_handle, [globus\\_ftp\\_control\\_callback\\_t](#) close\_callback\_func, void \*close\_arg)

#### 6.4.1 Detailed Description

FTP Data Connection Configuration and Management.

#### 6.4.2 Function Documentation

**6.4.2.1 globus\_result\_t globus\_ftp\_control\_data\_connect\_read (globus\_ftp\_control\_handle\_t \* *handle*, globus\_ftp\_control\_data\_connect\_callback\_t *callback*, void \* *user\_arg*)**

Create an incoming FTP data connection.

This function will register a globus\_io\_{accept, connect}. Further accepts/connects are done by registering a new accept/connect in the current accept/connect callback. A call to either [globus\\_ftp\\_control\\_local\\_pasv\(\)](#) or [globus\\_ftp\\_control\\_local\\_port\(\)](#) needs to precede this calling this function. This function may be followed by a globus\_ftp\_data\_read.

##### Parameters:

***handle*** A pointer to a FTP control handle which is configured to create an incoming data connection.

***callback*** This callback is called when the connection occurs. This parameter may be NULL.

***user\_arg*** The user argument passed to the connect callback.

**6.4.2.2 globus\_result\_t globus\_ftp\_control\_data\_set\_interface (globus\_ftp\_control\_handle\_t \* *handle*, const char \* *interface\_addr*)**

Create an outgoing FTP data connection.

This function sets the interface that will be used to send and receive information along the data channel.

**Parameters:**

*handle* A pointer to a FTP control handle which is configured to create an outgoing data connection.

*interface\_addr*

**6.4.2.3 globus\_result\_t globus\_ftp\_control\_data\_connect\_write (globus\_ftp\_control\_handle\_t \* *handle*, globus\_ftp\_control\_data\_connect\_callback\_t *callback*, void \* *user\_arg*)**

Create an outgoing FTP data connection.

This function will register a globus\_io\_{accept, connect}. Further accepts/connects are done by registering a new accept/connect in the current accept/connect callback. A call to either [globus\\_ftp\\_control\\_local\\_pasv\(\)](#) or [globus\\_ftp\\_control\\_local\\_port\(\)](#) needs to precede this calling this function. This function may be followed by a [globus\\_ftp\\_data\\_write](#).

**Parameters:**

*handle* A pointer to a FTP control handle which is configured to create an outgoing data connection.

*callback* This callback is called when the connection occurs. This parameter may be NULL.

*user\_arg* The user argument passed to the connect callback.

**6.4.2.4 globus\_result\_t globus\_ftp\_control\_data\_add\_channels (globus\_ftp\_control\_handle\_t \* *handle*, unsigned int *num\_channels*, unsigned int *stripe\_ndx*)**

Opens additional data channels (connections) to the host identified by the stripe parameter.

**Parameters:**

*handle* A pointer to a FTP control handle. This handle is used to determine the host corresponding to the stripe number and to store information about any channels added by this function.

*num\_channels* The number of additional channels to add.

*stripe\_ndx* A integer identifying the stripe to add channels too. In the case of non-striped transfer this parameter will be ignored.

**6.4.2.5 globus\_result\_t globus\_ftp\_control\_data\_send\_eof (globus\_ftp\_control\_handle\_t \* *handle*, int *count*[], int *array\_size*, globus\_bool\_t *eof\_message*, globus\_ftp\_control\_callback\_t *cb*, void \* *user\_arg*)**

Sends an eof message to each stripe along an open data connection.

**Parameters:**

*handle* A pointer to a FTP control handle. This handle contains the the state for a conneciton.

*count[]* This array of integers should contain an integer that will be added to the current parallel data connection count on each stripe. The order of the integers corresponds to each stripe in the same order as what was returned from [local\\_port\(\)](#).

An eof message must be sent to all receiving hosts in a transfer. The message contains the total number of data connections used by each stripe. Many stripes may be sending to a single receiver but only one eof message may be sent. The count parameter allows the user to pass in the total number of data connections used by all other hosts. The local values are added to the passed in values and then sent to the receiver.

**Parameters:**

*array\_size* The number of elements in *count[]*.

*eof\_message*

*cb* The function to be called when the eof message has been called.

*user\_arg* A user pointer that is threaded through to the user callback.



**6.4.2.6 globus\_result\_t globus\_ftp\_control\_data\_remove\_channels (globus\_ftp\_control\_handle\_t \* *handle*, unsigned int *num\_channels*, unsigned int *stripe\_ndx*)**

Removes data channels (connections) to the host identified by the stripe parameter.

**Parameters:**

*handle* A pointer to a FTP control handle. This handle is used to determine the host corresponding to the stripe number and to update information about any channels removed by this function.

*num\_channels* The number of channels to remove.

*stripe\_ndx* A integer identifying the stripe to remove channels from. In the case of non-striped transfer this parameter will be ignored.

**6.4.2.7 globus\_result\_t globus\_ftp\_control\_data\_query\_channels (globus\_ftp\_control\_handle\_t \* *handle*, unsigned int \* *num\_channels*, unsigned int *stripe\_ndx*)**

Returns the number of currently open channels for the host identified by the stripe parameter.

This number may be less then the level of parallelism specified in `local_parallelism`, due to the possibility that some channels have not yet connected.

**Parameters:**

*handle* A pointer to a FTP control handle. This handle is used to determine the host corresponding to "stripe" and number of channels corresponding to that host.

*num\_channels*

*stripe\_ndx* A integer identifying the stripe for which to return the number of channels. In the case of non-striped transfer this parameter should be zero.

**6.4.2.8 globus\_result\_t globus\_ftp\_control\_data\_get\_total\_data\_channels (globus\_ftp\_control\_handle\_t \* *handle*, unsigned int \* *num\_channels*, unsigned int *stripe\_ndx*)**

Returns the total number of data channels used so far in the current transfer on the given stripe.

**Parameters:**

*handle* A pointer to a FTP control handle. This handle is used to determine the host corresponding to "stripe" and number of channels corresponding to that host.

*num\_channels*

*stripe\_ndx* A integer identifying the stripe for which to return the number of channels. In the case of non-striped transfer this parameter should be zero.

**6.4.2.9 globus\_result\_t globus\_ftp\_control\_local\_send\_eof (globus\_ftp\_control\_handle\_t \* *handle*, globus\_bool\_t *send\_eof*)**

Determines if the library will automatically send an EOF message in extended block mode, or if the user will have to explicitly do it by calling [globus\\_ftp\\_control\\_data\\_send\\_eof\(\)](#).

**Parameters:**

*handle* The ftp handle you wish to sent the `send_eof` attribute on.

*send\_eof* A boolean representing whether or not to automatically send an EOF message.

**6.4.2.10 globus\_result\_t globus\_ftp\_control\_local\_parallelism (globus\_ftp\_control\_handle\_t \* *handle*, globus\_ftp\_control\_parallelism\_t \* *parallelism*)**

Set the parallelism information in a FTP control handle.

**Parameters:**

*handle* A pointer to the FTP control handle for which the parallelism information is to be updated

*parallelism* A structure containing parallelism information

**6.4.2.11 globus\_result\_t globus\_ftp\_control\_local\_pasv (globus\_ftp\_control\_handle\_t \* *handle*, globus\_ftp\_control\_host\_port\_t \* *address*)**

Create a local listening socket, bind it and return the address the socket is listening to.

If there is a existing data connection it is closed.

**Parameters:**

*handle* A pointer to a FTP control handle. Information about the listening socket is stored in the handle.

*address* The host IP address and port is returned through this parameter.

**6.4.2.12 globus\_result\_t globus\_ftp\_control\_local\_spas (globus\_ftp\_control\_handle\_t \* *handle*, globus\_ftp\_control\_host\_port\_t *addresses*[], unsigned int *num\_addresses*)**

Create num\_addresses local listening sockets, bind them and return the addresses the sockets are listening to.

If there is a existing data connection it is closed.

**Parameters:**

*handle* A pointer to a FTP control handle. Information about the listening sockets is stored in the handle.

*addresses* The host IP addresses and ports are returned through this parameter.

*num\_addresses* The number of listening sockets to create

**6.4.2.13 globus\_result\_t globus\_ftp\_control\_local\_port (globus\_ftp\_control\_handle\_t \* *handle*, globus\_ftp\_control\_host\_port\_t \* *address*)**

Insert the host/port information returned by a PASV on the remote host into the local FTP control handle.

(close any outstanding data con)

**Parameters:**

*handle* A pointer to the FTP control handle into which to insert the host/port information

*address* The host IP address and port

**6.4.2.14 globus\_result\_t globus\_ftp\_control\_local\_spor (globus\_ftp\_control\_handle\_t \* *handle*, globus\_ftp\_control\_host\_port\_t *addresses*[], unsigned int *num\_addresses*)**

Insert the host/port addresses returned by a SPAS on the remote host into the local FTP control handle.

If there are any outstanding data connections at this point, they are closed.

**Parameters:**

*handle* A pointer to the FTP control handle into which to insert the host/port addresses

*addresses* The host IP addresses and port numbers

*num\_addresses* The number of addresses

**6.4.2.15 globus\_result\_t globus\_ftp\_control\_local\_type (globus\_ftp\_control\_handle\_t \* *handle*, globus\_ftp\_control\_type\_t *type*, int *form\_code*)**

Update the FTP control handle with the given type information.

**Parameters:**

*handle* A pointer to the FTP control handle to be updated

*type* The type of the data connection. Possible values are: ASCII, EBCDIC, IMAGE and LOCAL. Currently only ASCII and IMAGE types are supported.

*form\_code* The logical byte size parameter for the LOCAL type.

**6.4.2.16 globus\_result\_t globus\_ftp\_control\_local\_mode (globus\_ftp\_control\_handle\_t \* *handle*, globus\_ftp\_control\_mode\_t *mode*)**

Update the FTP control handle with the given mode information.

**Parameters:**

*handle* A pointer to the FTP control handle to be updated

*mode* Specifies the mode of the data connection. Possible modes are STREAM, BLOCK, EXTENDED BLOCK and COMPRESSED. Out of these only STREAM and EXTENDED BLOCK are supported in this implementation. Also, EXTENDED BLOCK is only supported in combination with the IMAGE type.

**6.4.2.17 globus\_result\_t globus\_ftp\_control\_local\_tcp\_buffer (globus\_ftp\_control\_handle\_t \* *handle*, globus\_ftp\_control\_tcpbuffer\_t \* *tcp\_buffer*)**

Update the FTP control handle with the given socket buffer information.

**Parameters:**

*handle* A pointer to the FTP control handle to be updated

*tcp\_buffer* A pointer to the socket buffer.

**6.4.2.18 globus\_result\_t globus\_ftp\_control\_local\_dcau (globus\_ftp\_control\_handle\_t \* *handle*, const globus\_ftp\_control\_dcau\_t \* *dcau*, gss\_cred\_id\_t *delegated\_credential\_handle*)**

Update the FTP control handle with the given data channel authentication information.

If authentication is set to GLOBUS\_FTP\_CONTROL\_DCAU\_NONE, then protection will also be disabled for this control handle.

**Parameters:**

*handle* A pointer to the FTP control handle to be updated

*dcau* A parameter specifying the data channel authentication mode. Possible values are No Authentication, Self Authentication and Subject-name authentication.

*delegated\_credential\_handle*

**6.4.2.19 globus\_result\_t globus\_ftp\_control\_local\_pbsz (globus\_ftp\_control\_handle\_t \* *handle*, unsigned long *bufsize*)**

Update the FTP control handle with the given protection buffer size information.

This function sets protection buffer size to be used by this handle. This value is used to determine how much data will be sent in each packet during a protected data transfer.

**Parameters:**

*handle* A pointer to the FTP control handle to be updated

*bufsize* A parameter specifying the protection buffer size value.

**6.4.2.20 globus\_result\_t globus\_ftp\_control\_get\_pbsz (globus\_ftp\_control\_handle\_t \* *handle*, unsigned long \* *bufsize*)**

Query the FTP control handle for the protection buffer size information.

This function queries the handle to determine the protection buffer size which is used by this handle. This value is used to determine how much data will be sent in each packet during a protected data transfer.

**Parameters:**

*handle* A pointer to the FTP control handle to be updated

*bufsize* A pointer to a parameter to store the value of the protection buffer size.

**6.4.2.21 globus\_result\_t globus\_ftp\_control\_local\_stru (globus\_ftp\_control\_handle\_t \* *handle*, globus\_ftp\_control\_structure\_t *structure*)**

Updates the handle with information on the structure of the data being sent on the data channel.

This function updates the handle with the provided structure information. At this point the only structure type that is supported is the file type.

**Parameters:**

*handle* A pointer to a FTP control handle. The handle contains information about the current state of the control and data connections.

*structure* This parameter is used to pass the structure information. Possible values are file, record and page. Only the file type is supported

**6.4.2.22 globus\_result\_t globus\_ftp\_control\_data\_write (globus\_ftp\_control\_handle\_t \* *handle*, globus\_byte\_t \* *buffer*, globus\_size\_t *length*, globus\_off\_t *offset*, globus\_bool\_t *eof*, globus\_ftp\_control\_data\_callback\_t *callback*, void \* *callback\_arg*)**

Writes data from the supplied buffer to data connection(s).

This function writes contained in the buffer to the data channel(s).

**Parameters:**

*handle* A pointer to a FTP control handle. The handle contains information about the current state of the control and data connections.

*buffer* A user supplied buffer from which data will written to the data connection(s)

*length* The length of the data contained in the buffer.

*offset* The offset in the file at which the data in the buffer starts

*eof* Indicates that the buffer is that last part of a file. In the striped case this will cause a EOF block to be send to every data node involved in the transfer.

*callback* The function to be called once the data has been sent

*callback\_arg* User supplied argument to the callback function

**6.4.2.23** `globus_result_t globus_ftp_control_data_read (globus_ftp_control_handle_t * handle, globus_byte_t * buffer, globus_size_t max_length, globus_ftp_control_data_callback_t callback, void * callback_arg)`

Reads data from data connection(s) and put them in the supplied buffer.

This function takes the given buffer and will try to read data from the data connection(s).

**Parameters:**

***handle*** A pointer to a FTP control handle. The handle contains information about the current state of the control and data connections.

***buffer*** A user supplied buffer into which data from the data connection(s) will be written

***max\_length*** The maximum length of the data that can be written to the buffer

***callback*** The function to be called once the data has been read

***callback\_arg*** User supplied argument to the callback function

**6.4.2.24** `globus_result_t globus_ftp_control_local_layout (globus_ftp_control_handle_t * handle, globus_ftp_control_layout_t * layout, globus_size_t data_size)`

Update the handle with the layout and the size of the data sent over the data channel.

This function is deprecated. The interface will be the changed to that of `globus_X_ftp_control_local_layout()`

**Parameters:**

***handle*** A pointer to the FTP control handle into which to insert the layout information.

***layout*** A variable containing the layout information

***data\_size*** The size of the data that is going to be sent. This may be needed to interpret the layout information.

**6.4.2.25** `globus_result_t globus_ftp_control_create_data_info (globus_ftp_control_handle_t * handle, globus_ftp_control_data_write_info_t * data_info, globus_byte_t * buffer, globus_size_t length, globus_off_t offset, globus_bool_t eof, globus_ftp_control_data_callback_t callback, void * callback_arg)`

Create a `globus_ftp_control_data_write_info_t` structure.

This function populates a `globus_ftp_control_data_callback_t` structure with valid information. This structure provides the user a way to register several data writes with a single callback. This is quite useful to the writer of enqueue functions. It allows a single call to `globus_ftp_control_data_write()` to be broken up into many writes, potentially on different stripes, and for a single callback to be called when all are finished.

**Parameters:**

***handle*** A pointer to a FTP control handle. The handle contains information about the current state of the control and data connections.

***data\_info*** The `globus_ftp_control_data_write_info_t` structure to be released.

***buffer*** The pointer to the user buffer that will be passed to the callback argument when there are zero references to `data_info`. This is intended to be the start of all the data the user intends to write using `globus_ftp_control_data_write_stripe()`, but it does not have to be.

***length*** The length of the memory segment pointed to by the argument `buffer`.

***offset*** The file offset of the data segment specified.

***eof*** This should be set to true if the user plans on registering eof on the `data_info` structure.

***callback*** The user function to be called when all references to `data_info` are released. This occurs after all data registered for write from `globus_ftp_control_data_write_stripe` have occurred and the user calls `globus_ftp_control_release_data_info()`. The callback is passed all of the arguments passed to this function with the exception of `data_info`.

***callback\_arg*** User supplied argument to the callback function

**6.4.2.26** `globus_result_t globus_ftp_control_release_data_info (globus_ftp_control_handle_t * handle, globus_ftp_control_data_write_info_t * data_info)`

Release a `data_info` structure.

This function releases all memory and references created when a call to `globus_ftp_control_create_data_info()` was made. For every call to `globus_ftp_control_create_data_info()` a call to this function must be made.

**Parameters:**

*handle* A pointer to a FTP control handle. The handle contains information about the current state of the control and data connections.

*data\_info* The `globus_ftp_control_data_write_info_t` structure to be released.

**6.4.2.27** `globus_result_t globus_ftp_control_data_write_stripe (globus_ftp_control_handle_t * handle, globus_byte_t * buffer, globus_size_t length, globus_off_t offset, globus_bool_t eof, int stripe_ndx, globus_ftp_control_data_callback_t callback, void * callback_arg)`

Write FTP data to a particular stripe.

This function allows the user to write to a specified stripe. The stripe index relates to the order passed into `local_spor()`. This function differs from `globus_ftp_control_data_write()` in that no enqueue function is needed since the user specifies the stripe on which data is written. In order to use this function the user must have a valid pointer to a `globus_ftp_control_data_write_info_t` structure. The `data_info` structure can be obtained by a call to `globus_ftp_control_create_data_info()`. Many calls to this function can be made, but only a single user callback occurs per creation of a `globus_ftp_control_data_write_info_t` structure.

**Parameters:**

*handle* A pointer to a FTP control handle. The handle contains information about the current state of the control and data connections.

*buffer* a pointer to the data the user wishes to send along the FTP data channels.

*length* the length of the data pointer to by the parameter `buffer`.

*offset* the offset into the file of the data.

*eof* A boolean stating that this will be the last chunk of data registered on the given stripe. In order to properly send an eof message the user must register an eof on every stripe.

*stripe\_ndx* The index of the stripe on which the data will be sent. The index of each stripe is determined by the call to `local_spor` or `local_spor`.

*callback* The function to be called once the data has been sent

*callback\_arg* User supplied argument to the callback function

**6.4.2.28** `globus_result_t globus_X_ftp_control_data_write_stripe (globus_ftp_control_handle_t * handle, globus_byte_t * buffer, globus_size_t length, globus_off_t offset, globus_bool_t eof, int stripe_ndx, globus_ftp_control_data_write_info_t * data_info)`

Write data on a specific stripe from an enqueue callback function only.

This function allows the user to register the write of ftp data on a specific stripe. This function can only be called from an enqueue function callback. This function should be used only by the implementor of an enqueue function. It should be viewed as unstable and used only by advanced users. This is the only function in the library that the enqueue function implementor is allowed from the enqueue callback.

**Parameters:**

*handle* A pointer to a FTP control handle. The handle contains information about the current state of the control and data connections.

**buffer** a pointer to the data the user wishes to send along the FTP data channels.

**length** the length of the data pointer to by the parameter buffer.

**offset** the offset into the file of the data.

**eof** a boolean stating that this is the last buffer to be registered. When using the `_X_` version of this function the user does not need to register an eof on each stripe, the control library will take care of that internally.

**stripe\_ndx** The index of the stripe on which the data will be sent. The index of each stripe is determined by the call to `local_spas` or `local_spor`.

**data\_info** An opaque structure that is passed into the enqueue function and contains reference count and state information. The same `data_info` pointer that is passed into the enqueue function must be used for this parameter.

#### 6.4.2.29 `globus_result_t globus_ftp_control_data_force_close (globus_ftp_control_handle_t * control_handle, globus_ftp_control_callback_t close_callback_func, void * close_arg)`

Forces an immediate close of all data connections.

##### Parameters:

**control\_handle** The `globus_ftp_control_handle_t` that has its data connections closed.

**close\_callback\_func** A user function that will be called when the data connections are closed.

**close\_arg** The user argument that will be threaded through to `close_callback_func`.

## 6.5 globus\_ftp\_control\_layout.c File Reference

### 6.5.1 Detailed Description

## 6.6 globus\_ftp\_control\_server.c File Reference

FTP Server-side Control Connection Management.

### Functions

- `globus_result_t globus_ftp_control_server_handle_init (globus_ftp_control_server_t *handle)`
- `globus_result_t globus_ftp_control_server_handle_destroy (globus_ftp_control_server_t *handle)`
- `globus_result_t globus_ftp_control_server_listen (globus_ftp_control_server_t *server_handle, unsigned short *port, globus_ftp_control_server_callback_t callback, void *callback_arg)`
- `globus_result_t globus_ftp_control_command_init (globus_ftp_control_command_t *command, char *raw_command, globus_ftp_control_auth_info_t *auth_info)`
- `globus_result_t globus_ftp_control_command_destroy (globus_ftp_control_command_t *command)`
- `globus_result_t globus_ftp_control_command_copy (globus_ftp_control_command_t *dest, globus_ftp_control_command_t *src)`
- `globus_result_t globus_ftp_control_server_stop (globus_ftp_control_server_t *listener, globus_ftp_control_server_callback_t callback, void *callback_arg)`
- `globus_result_t globus_ftp_control_server_accept (globus_ftp_control_server_t *listener, globus_ftp_control_handle_t *handle, globus_ftp_control_callback_t callback, void *callback_arg)`
- `globus_result_t globus_ftp_control_server_authenticate (globus_ftp_control_handle_t *handle, globus_ftp_control_auth_requirements_t auth_requirements, globus_ftp_control_auth_callback_t callback, void *callback_arg)`
- `globus_result_t globus_ftp_control_read_commands (globus_ftp_control_handle_t *handle, globus_ftp_control_command_callback_t callback, void *callback_arg)`
- `globus_result_t globus_ftp_control_send_response (globus_ftp_control_handle_t *handle, const char *resp_spec, globus_ftp_control_callback_t callback, void *callback_arg,...)`

### 6.6.1 Detailed Description

FTP Server-side Control Connection Management.

### 6.6.2 Function Documentation

#### 6.6.2.1 `globus_result_t globus_ftp_control_server_handle_init (globus_ftp_control_server_t * handle)`

Initialize a globus ftp server handle.

This function will set up (i.e. initialize all mutexes and variables) a globus ftp server handle. It will also enter the handle in a list used by the module activation/deactivation functions.

**Parameters:**

*handle* The handle to initialize.

**Returns:**

- GLOBUS\_SUCCESS
- invalid handle

#### 6.6.2.2 `globus_result_t globus_ftp_control_server_handle_destroy (globus_ftp_control_server_t * handle)`

Destroy a globus ftp server handle.

This function will free up all dynamically allocated memory associated with a given globus ftp server handle. It will also remove the handle from a list used by the module activation/deactivation functions. This function should only be called after a call to `globus_ftp_control_server_stop`.

**Parameters:**

*handle* The handle to destroy.

**Returns:**

- success
- invalid handle
- handle is still in listening state

#### 6.6.2.3 `globus_result_t globus_ftp_control_server_listen (globus_ftp_control_server_t * server_handle, unsigned short * port, globus_ftp_control_server_callback_t callback, void * callback_arg)`

Start listening on a given port for FTP client connections.

This function starts the listening on *\*port* for connections from ftp clients. When a connection request is made *callback* is called and passed *callback\_arg*. Upon return from this function the *server\_handle* structure is initialized.

**Parameters:**

*server\_handle* A pointer to a initialized server handle.

*port* A pointer to the port to listen on. If the initial value is zero it will be set to the default value.

*callback* The callback function called when connection requests are made.

*callback\_arg* The user argument passed to the callback function when connection requests are made.

**Note:**

I'm not providing any mechanism for making sure that this function is only called once. Is this needed?



#### 6.6.2.4 `globus_result_t globus_ftp_control_command_init (globus_ftp_control_command_t * command, char * raw_command, globus_ftp_control_auth_info_t * auth_info)`

Initialize a command structure.

This function initializes a command structure based on a null terminated string representing one line of input from the client. The command structure is used as a convenience to determine what command the client issued. This function parses a command string sent by a client and populates the command argument appropriately. In the GSSAPI case it will also decode and unwrap the command before parsing it.

##### Parameters:

*command* A pointer to the command structure to be initialized

*raw\_command* A null terminated line of client input. Should contain one command.

*auth\_info* Authentication information needed for unwrapping a command

#### 6.6.2.5 `globus_result_t globus_ftp_control_command_destroy (globus_ftp_control_command_t * command)`

Destroy a command structure.

This function frees up the memory allocated to the command argument.

##### Parameters:

*command* The command structure whose associated memory is to be freed

#### 6.6.2.6 `globus_result_t globus_ftp_control_command_copy (globus_ftp_control_command_t * dest, globus_ftp_control_command_t * src)`

Creates a copy of a command structure.

This function should be called when the user needs to make a copy of a command structure.

##### Parameters:

*dest* The area of memory that the command structure is copied to.

*src* The command structure to be copied.

#### 6.6.2.7 `globus_result_t globus_ftp_control_server_stop (globus_ftp_control_server_t * listener, globus_ftp_control_server_callback_t callback, void * callback_arg)`

Stop the GSIFTP server from listening for client connections.

This function stops listening on the given listener object for client connections. All existing client connections are left open.

##### Parameters:

*listener* the `globus_ftp_control_server_t` object that should no longer listen for connections.

*callback* The user callback that will be called when the server structure is no longer listening.

*callback\_arg* The user argument that is passed into callback.

**6.6.2.8 globus\_result\_t globus\_ftp\_control\_server\_accept (globus\_ftp\_control\_server\_t \* listener, globus\_ftp\_control\_handle\_t \* handle, globus\_ftp\_control\_callback\_t callback, void \* callback\_arg)**

Accept a client connection request.

This function is called to accept a connection request from a client.

When the listen callback is called (see globus\_ftp\_control\_server\_listen) a client has requested a connection. This function must be called to accept that user connection request. Once the connection is established or if a error occurs, the callback function is called.

**Parameters:**

*listener* The server object that received the connection request.

*handle* The control connection object. This structure will be populated and passed to the callback when the client is authorized. This structure represents the control connection between the server and client. It will be used to read commands from the client and send responses to the client.]

*callback* The function called when the client connection has been accepted.

*callback\_arg* The user argument passed to the callback.

**Note:**

This functions assumes the the server and control handles have been initialized prior to calling this function.

**6.6.2.9 globus\_result\_t globus\_ftp\_control\_server\_authenticate (globus\_ftp\_control\_handle\_t \* handle, globus\_ftp\_control\_auth\_requirements\_t auth\_requirements, globus\_ftp\_control\_auth\_callback\_t callback, void \* callback\_arg)**

Authenticate a client connection.

This function is called to authenticate a connection from a client.

After a client connection has been accepted (using the globus\_ftp\_control\_server\_accept call), this function should be called to authenticate the client. The caller of this function may specify certain authentication requirements using the auth\_requirements parameter.

**Parameters:**

*handle* The control connection object. This structure will be populated and passed to the callback when the client is authorized. This structure represents the control connection between the server and client. It will be used to read commands from the client and send responses to the client.]

*auth\_requirements* This structure represents the authentication requirements that the user has for a given connection. For example GSIFTP user name, password, and account.

*callback* The function called when the client authentication has been accepted or rejected.

*callback\_arg* The user argument passed to the callback.

**Note:**

It is up to the user of this function to send the reply to the last command of the authentication sequence.

This functions assumes the the server and control handles have been initialized prior to calling this function.

**6.6.2.10 globus\_result\_t globus\_ftp\_control\_read\_commands (globus\_ftp\_control\_handle\_t \* handle, globus\_ftp\_control\_command\_callback\_t callback, void \* callback\_arg)**

Begin reading GSIFTP commands on a given control connection.

This function begins reading control commands on a globus\_ftp\_control\_handle\_t. When a command is read the callback function is called.

**Parameters:**

*handle* The control connection handle that commands will be read from. Prior to calling this the function `globus_ftp_control_handle_t` must be populated via a call to `globus_ftp_control_accept()`.

*callback* The user callback that will be called when commands are read.

*callback\_arg* The user argument passed to the callback.

**6.6.2.11** `globus_result_t globus_ftp_control_send_response (globus_ftp_control_handle_t * handle, const char * respspec, globus\_ftp\_control\_callback\_t callback, void * callback_arg, ...)`

Send a response to the GSIFTP client.

This function sends a GSIFTP formatted response to the client. When a command callback is received the user calls this function to respond to the clients request.

**Parameters:**

*handle* The control connection to send the response across.

*respspec* A formatted string representing the users response.

*callback* The user callback that will be called when the response has been sent.

*callback\_arg* The user argument passed to the callback.

## 7 globus ftp control Page Documentation

### 7.1 GridFTP: Protocol Extensions to FTP for the Grid

#### 7.1.1 Introduction

This section defines extensions to the FTP specification STD 9, RFC 959, [FILE TRANSFER PROTOCOL \(FTP\)](#) (October 1985) These extensions provide striped data transfer, parallel data transfer, extended data transfer, data buffer size configuration, and data channel authentication.

The following new commands are introduced in this specification

- [Striped Passive \(SPAS\)](#)
- [Striped Data Port \(SPOR\)](#)
- [Extended Retrieve \(ERET\)](#)
- [Extended Store \(ESTO\)](#)
- [Set Data Buffer Size \(SBUF\)](#)
- [Data Channel Authentication Mode \(DCAU\)](#)

A new transfer mode ([extended-block mode](#)) is introduced for parallel and striped data transfers. Also, a set of extension [options to RETR](#) are added to control striped data layout and parallelism.

The following new feature names are to be included in the FTP server's response to FEAT if it implements the following sets of functionality

#### PARALLEL

The server supports the SPOR, SPAS, the RETR options mentioned above, and extended block mode.

#### ESTO

The server implements the ESTO command as described in this document.

**ERET**

The server implements the ERET command as described in this document.

**SBUF**

The server implements the SBUF command as described in this document.

**DCAU**

The server implements the DCAU command as described in this document, including the requirement that data channels are authenticated by default, if [RFC 2228](#) authentication is used to establish the control channel.

### 7.1.2 Terminology

**Parallel transfer**

From a single data server, splitting file data for transfer over multiple data connections.

**Striped transfer**

Distributing a file's data over multiple independent data nodes, and transferring over multiple data connections.

**Data Node**

In a striped data transfer, a data node is one of the stripe destinations returned in the SPAS command, or one of the stripe destinations sent in the SPOR command.

**DTP**

The data transfer process establishes and manages the data connection. The DTP can be passive or active.

**PI**

The protocol interpreter. The user and server sides of the protocol have distinct roles implemented in a user-PI and a server-PI.

### 7.1.3 FTP Standards Used

- RFC 959, [FILE TRANSFER PROTOCOL \(FTP\)](#), J. Postel, R. Reynolds (October 1985)
  - Commands used by GridFTP
    - \* USER
    - \* PASS
    - \* ACCT
    - \* CWD
    - \* CDUP
    - \* QUIT
    - \* REIN
    - \* PORT
    - \* PASV
    - \* TYPE
    - \* MODE
    - \* RETR
    - \* STOR
    - \* STOU
    - \* APPE
    - \* ALLO
    - \* REST
    - \* RNFR

- \* RNTD
- \* ABOR
- \* DELE
- \* RMD
- \* MKD
- \* PWD
- \* LIST
- \* NLST
- \* SITE
- \* SYST
- \* STAT
- \* HELP
- \* NOOP
- Features used by GridFTP
  - \* ASCII and Image types
  - \* Stream mode
  - \* File structure
- RFC 2228, [FTP Security Extensions](#), Horowitz, M. and S. Lunt (October 1997)
  - Commands used by GridFTP
    - \* AUTH
    - \* ADAT
    - \* MIC
    - \* CONF
    - \* ENC
  - Features used by GridFTP
    - \* GSSAPI authentication
- RFC 2389, [Feature negotiation mechanism for the File Transfer Protocol](#), P. Hethmon , R. Elz (August 1998)
  - Commands used by GridFTP
    - \* FEAT
    - \* OPTS
  - Features used by GridFTP
- [FTP Extensions](#), R. Elz, P. Hethmon (September 2000)
  - Commands used by GridFTP
    - \* SIZE
  - Features used by GridFTP
    - \* Restart of a stream mode transfer

### 7.1.4 Striped Passive (SPAS)

This extension is used to establish a vector of data socket listeners for a server with one or more stripes. This command **MUST** be used in conjunction with the extended block mode. The response to this command includes a list of host and port addresses the server is listening on.

Due to the nature of the extended block mode protocol, SPAS must be used in conjunction with data transfer commands which receive data (such as STOR, ESTO, or APPE) and can not be used with commands which send data on the data channels.

#### Syntax

The syntax of the SPAS command is:

```
spas = "SPAS" <CRLF>
```

#### Responses

The server-PI will respond to the SPAS command with a 229 reply giving the list of host-port strings for the remote server-DTP or user-DTP to connect to.

```
spas-response = "229-Entering Striped Passive Mode" CRLF
                1*(<SP> host-port CRLF)
                229 End
```

Where the command is correctly parsed, but the server-DTP cannot process the SPAS request, it must return the same error responses as the PASV command.

#### OPTS for SPAS

There are no options in this SPAS specification, and hence there is no OPTS command defined.

### 7.1.5 Striped Data Port (SPOR)

This extension is to be used as a complement to the SPAS command to implement striped third-party transfers. This command **MUST** always be used in conjunction with the extended block mode. The argument to SPOR is a vector of host/TCP listener port pairs to which the server is to connect. This

Due to the nature of the extended block mode protocol, SPOR must be used in conjunction with data transfer commands which send data (such as RETR, ERET, LIST, or NLST) and can not be used with commands which receive data on the data channels.

#### Syntax

The syntax of the SPOR command is:

```
SPOR 1*(<SP> <host-port>) <CRLF>
```

The host-port sequence in the command structure **MUST** match the host-port replies to a SPAS command.

#### Responses

The server-PI will respond to the SPOR command with the same response set as the PORT command described in the [ftp specification](#).

## OPTS for SPOR

There are no options in this SPOR specification, and hence there is no OPTS command defined.

### 7.1.6 Extended Retrieve (ERET)

The extended retrieve extension is used to request that a retrieve be done with some additional processing on the server. This command is an extensible way of providing server-side data reduction or other modifications to the RETR command. This command is used in place of OPTS to the RETR command to allow server side processing to be done with a single round trip (one command sent to the server instead of two) for latency-critical applications.

ERET may be used with either the data transports defined in RFC 959, or using extended block mode as defined in this document. Using an ERET creates a new virtual file which will be sent, with its own size and byte range starting at zero. Restart markers generated while processing an ERET are relative to the beginning of this view of the file.

#### Syntax

The syntax of the ERET command is

```
ERET <SP> <retrieve-mode> <SP> <filename>

retrieve-mode ::= P <SP> <offset> <SP> <size>
offset ::= 64 bit integer
size ::= 64 bit integer
```

The **retrieve-mode** defines behavior of the extended-retrieve mode. There is one mode defined by this specification, but other general purpose or application-specific ones may be added later.

#### modes\_ERET Extended Retrieve Modes

##### Partial Retrieve Mode (P)

A section of the file will be retrieved from the data server. The section is defined by the starting **offset** and extent **size** parameters. When used with extended block mode, the extended block headers sent along with data will send the data with offset of 0 meaning the beginning of the section of the file which was requested.

### 7.1.7 Extended Store (ESTO)

The extended store extension is used to request that a store be done with some additional processing on the server. Arbitrary data processing algorithms may be added by defining additional ESTO store-modes. Similar to the ERET, the ESTO command expects data sent to satisfy the request to be sent as if it were a new file with data block offset 0 being beginning the beginning of the new file.

The format of the ESTO command is

```
ESTO <SP> <store-mode> <filename>

store-mode ::= A <SP> <offset>
```

The store-mode defines the behavior of the extended store. There is one mode defined by this specification, but others may be added later.

## Extended Store Modes

### Adjusted store (A)

The data in the file is to stored with **offset** added to the file pointer before storing the blocks of the file. In extended block mode, this value is added to the offset in the extended block header by the server when writing to disk. Extended block headers should therefore send the beginning of the byte range on the data channel with offset of zero. In stream mode, the offset is added to the implicit offset of 0 for the beginning of the data before writing. If a stream mode restart marker is used in conjunction with this ESTO mode, the restart marker's offset is added to the offset passed as the parameter to the adjusted store.

### 7.1.8 Set Buffer Size (SBUF)

This extension adds the capability of a client to set the TCP buffer size for subsequent data connections to a value. This replaces the server-specific commands SITE RBUFSIZE, SITE RETRBUFSIZE, SITE RBUFSZ, SITE SBUFSIZE, SITE SBUFSZ, and SITE BUFSIZE. Clients may wish to consider supporting these other commands to ensure wider compatibility.

#### Syntax

The syntax of the SBUF command is

```
sbuf = SBUF <SP> <buffer-size>
buffer-size ::= <number>
```

The **buffer-size** value is the TCP buffer size in bytes. The TCP window size should be set accordingly by the server.

#### Response Codes

If the server-PI is able to set the buffer size state to the requested **buffer-size**, then it will return a 200 reply.

#### Note:

Even if the SBUF is accepted by the server, an error may occur later when the data connections are actually created, depending on how the server or client operating systems' TCP implementations.

### 7.1.9 Data Channel Authentication (DCAU)

This extension provides a method for specifying the type of authentication to be performed on FTP data channels. This extension may only be used when the control connection was authenticated using RFC 2228 Security extensions.

The format of the DCAU command is

```
DCAU <SP> <authentication-mode> <CRLF>
authentication-mode ::= <no-authentication>
```



```

| <authenticate-with-self>
| <authenticate-with-subject>

no-authentication ::= N
authenticate-with-self ::= A
authenticate-with-subject ::= S <subject-name>

subject-name ::= string

```

## Authentication Modes

- No authentication (**N**)  
No authentication handshake will be done upon data connection establishment.
- Self authentication (**A**)  
A security-protocol specific authentication will be used on the data channel. The identity of the remote data connection will be the same as the identity of the user which authenticated to the control connection.
- Subject-name authentication (**S**)  
A security-protocol specific authentication will be used on the data channel. The identity of the remote data connection **MUST** match the supplied **subject-name** string.

The default data channel authentication mode is **A** for FTP sessions which are RFC 2228 authenticated—the client must explicitly send a DCAU N message to disable it if it does not implement data channel authentication.

If the security handshake fails, the server should return the error response 432 (Data channel authentication failed).

### 7.1.10 Extended Block Mode

The striped and parallel data transfer methods described above require an extended transfer mode to support out-of-sequence data delivery, and partial data transmission per data connection. The extended block mode described here extends the block mode header to provide support for these as well as large blocks, and end-of-data synchronization.

Clients indicate that they want to use extended block mode by sending the command

```
MODE <SP> E <CRLF>
```

on the control channel before a transfer command is sent.

The structure of the extended block header is

Extended Block Header

```

+-----+-----/-----+-----/-----+
| Descriptor |   Byte Count   |   Offset Count   |
|   8 bits   |   64 bits     |   64 bits       |
+-----+-----/-----+-----/-----+

```

The descriptor codes are indicated by bit flags in the descriptor byte. Six codes have been assigned, where each code number is the decimal value of the corresponding bit in the byte.

Code	Meaning
128	End of data block is EOR (Legacy)
64	End of data block is EOF
32	Suspected errors in data block
16	Data block is a restart marker
8	End of data block is EOD for a parallel/striped transfer
4	Sender will close the data connection

With this encoding, more than one descriptor coded condition may exist for a particular block. As many bits as necessary may be flagged.

Some additional protocol is added to the extended block mode data channels, to properly handle end-of-file detection in the presence of an unknown number of data streams.

- When no more data is to be sent on the data channel, then the sender will mark the last block, or send a zero-length block after the last block with the EOD bit (8) set in the extended block header.
- After receiving an EOD the data connection can be cached for use in a subsequent transfer. To signify that the data connection will be closed the sender sets the close bit (4) in the header on the last message sent.
- The sender communicates end of file by sending an EOF message to all servers receiving data. The EOF message format follows.

#### Extended Block EOF Header

+-----+-----+-----+-----+		+-----+-----+-----+-----+	
Descriptor	unused	EOD count expected	
8 bits	64 bits	64 bits	
+-----+-----+-----+-----+		+-----+-----+-----+-----+	

EOF Descriptor. The EOF header descriptor has the same definition as the regular data message header described above.

EOD Count Expected. This 64 bit field represents the total number of data connections that will be established with the server receiving the file. This number is used by the receiver to determine it has received all of the data. When the number of EOD messages received equals the number represented by the "EOD Count Expected" field the receiver has hit end of file.

Simply waiting for EOD on all open data connections is not sufficient. It is possible that the receiver reads an EOD message on all of its open data connects while an additional data connection is in flight. If the receiver were to assume it reached end of file it would fail to receive the data on the in flight connection.

To handle EOF in the multi-striped server case a 126 response has been introduced. When receiving data from a striped server a client makes a control connection to a single host, but several host may create several data connections back to the client. Each host can independently decide how many data connections it will use, but only a single EOF message may be sent to back to the client, therefore it must be possible to aggregate the total number of data connections used in the transfer across the stripes. The 126 response serves this purpose.

The 126 is an intermediate response to RETR command. It has the following format.

"126" 1\*(count of data connections)

Several "Count of data connections" can be in a single reply. They correspond to the stripes returned in the response to the SPAS command.

Discussion of protocol change to enable bidirectional data channels brought up the following problem if doing bidirectional data channels

If the client is pasv, and sending to a multi-stripe server, then the server creates data connections connections; since the client didn't do SPAS, it cannot associate HOST/PORT pairs on the data connections with stripes on the server (it doesn't even know how many there are). it cannot reliably determine which nodes to send data to. (Becomes even more complex in the third-party transfer case, because the sender may have multiple stripes of data.) The basic problem is that we need to know logical stripe numbers to know where to send the data.

#### EOF Handling in Extended Block Mode

If you are in either striped or parallel mode, you will get exactly one EOF on each SPAS-specified ports (stripes). Hosts in extended block mode must be prepared to accept an arbitrary number of connections on each SPOR port before the EOF block is sent.

## Restarting

In general, opaque restart markers passed via the block header should not be used in extended block mode. Instead, the destination server should send extended data marker responses over the control connection, in the following form:

```
extended-mark-response = "111" <SP> "Range Marker" <SP> <byte-ranges-list>

byte-ranges-list       = <byte-range> [ *(", " <byte-range>) ]
byte-range              = <start-offset> "-" <end-offset>

start-offset            ::= <number>
end-offset              ::= <number>
```

The byte ranges in the marker are an incremental set of byte ranges which have been stored to disk by the data server. The complete restart marker is a concatenation of all byte ranges received by the client in 111 responses.

The client MAY combine adjacent ranges received over several range responses into any number of ranges when sending the REST command to the server to restart a transfer.

For example, the client, on receiving the responses:

```
111 Range Marker 0-29
111 Range Marker 30-89
```

may send, equivalently,

```
REST 0-29,30-89
REST 0-89
REST 30-59,0-29,60-89
```

to restart the transfer after those 90 bytes have been received.

The server MAY indicate that a given range of data has been received in multiple subsequent range markers. The client MUST be able to handle this. For example:

```
111 Range Marker 30-59
111 Range Marker 0-89
```

is equivalent to

```
111 Range Marker 30-59
111 Range Marker 0-29,60-89
```

Similarly, the client, if it is doing no processing of the restart markers, MAY send redundant information in a restart.

*Should these be allowed as restart markers for stream mode?*

## Performance Monitoring

In order to monitor the performance of extended block mode transfer, an additional preliminary reply MAY be transmitted over the control channel. This reply is of the form:

```
extended-perf-response = "112-Perf Marker" CRLF
                        <SP> "Timestamp:" <SP> <timestamp> CRLF
                        <SP> "Stripe Index:" <SP> <stripe-number> CRLF
```

```

<SP> "Stripe Bytes Transferred:" <SP> <byte count> CRLF
<SP> "Total Stripe Count:" <SP> <stripe count> CRLF
"112 End" CRLF

```

```
timestamp = <number> [ "." <digit> ]
```

is seconds since the epoch

The performance marker can contain these or any other perf-line facts which provide useful information about the current performance.

All perf-line facts represent an instantaneous state of the transfer at the given timestamp. The meaning of the facts are

- Timestamp - The time at which the server computed the performance information. This is in seconds since the epoch (00:00:00 UTC, January 1, 1970).
- Stripe Index - the index (0-number of stripes on the STOR side of the transfer) which this marker pertains to.
- Stripe Bytes Transferred - The number of bytes which have been received on this stripe.

A transfer start time can be specified by a perf marker with 'Stripe Bytes Transferred' set to zero. Only the first marker per stripe can be used to specify the start time of that stripe. Any subsequent markers with 'Stripe Bytes Transferred' set to zero simply indicates no data transfer over the interval.

A server should send a 'start' marker for each stripe. A server should also send a final perf marker for each stripe. This is a marker with 'Stripe Bytes Transferred' set to the total transfer size for that stripe.

### 7.1.11 Options to RETR

The options described in this section provide a means to convey striping and transfer parallelism information to the server-DTP. For the RETR command, the Client-FTP may specify a parallelism and striping mode it wishes the server-DTP to use. These options are only used by the server-DTP if the retrieve operation is done in extended block mode. These options are implemented as [RFC 2389](#) extensions.

The format of the RETR OPTS is specified by:

```

retr-opts      = "OPTS" <SP> "RETR" [ <SP> option-list ] CRLF
option-list    = [ layout-opts ";" ] [ parallel-opts ";" ]
layout-opts    = "StripeLayout=Partitioned"
                | "StripeLayout=Blocked;BlockSize=" <block-size>
parallel-opts  = "Parallelism=" <starting-parallelism> ","
                <minimum-parallelism> ","
                <maximum-parallelism>

block-size     ::= <number>
starting-parallelism ::= <number>
minimum-parallelism  ::= <number>
maximum-parallelism  ::= <number>

```

### Layout Options

The layout option is used by the source data node to send sections of the data file to the appropriate destination stripe. The various StripeLayout parameters are to be implemented as follows:

#### Partitioned

A partitioned data layout is one where the data is distributed evenly on the destination data nodes. Only one contiguous section of data is stored on each data node. A data node is defined here a single host-port mentioned in the SPOR command

## Blocked

A blocked data layout is one where the data is distributed in round-robin fashion over the destination data nodes. The data distribution is ordered by the order of the host-port specifications in the SPOR command. The **block-size** defines the size of blocks to be distributed.

## PLVL Parallelism Options

The parallelism option is used by the source data node to control how many parallel data connections may be established to each destination data node. This extension option provides for both a fixed level of parallelism, and for adapting the parallelism to the host/network connection, within a range. If the **starting-parallelism** option is set, then the server-DTP will make **starting-parallelism** connections to each destination data node. If the **minimum-parallelism** option is set, then the server may reduce the number of parallel connections per destination data node to this value. If the **maximum-parallelism** option is set, then the server may increase the number of parallel connections to per destination data node to at most this value.

### 7.1.12 References

- [1] Postel, J. and Reynolds, J., "**FILE TRANSFER PROTOCOL (FTP)**", STD 9, RFC 959, October 1985.
- [2] Hethmon, P. and Elz, R., "**Feature negotiation mechanism for the File Transfer Protocol**", RFC 2389, August 1998.
- [3] Horowitz, M. and Lunt, S., "**FTP Security Extensions**", RFC 2228, October 1997.
- [4] Elz, R. and Hethom, P., "**FTP Extensions**", IETF Draft, May 2001.

### 7.1.13 Appendix I: Implementation under GSI

There are several security components in this document which are extensions to the behavior of RFC 2228. These appendix attempts to clarify the protocol how these extensions map to the OpenSSL-based implementation of the GSSAPI known as GSI (Grid Security Infrastructure).

A client implementation which communicates with a server which supports the DCAU extension should delegate a limited credential set (using the GSS\_C\_DELEG\_FLAG and GSS\_C\_GLOBUS\_LIMITED\_DELEG\_PROXY\_FLAG flags to gss\_init\_sec\_context()). If delegation is not performed, the client MUST request that DCAU be disabled by requesting DCAU N, or the server will be unable to perform the default of DCAU A as described by this document.

When DCAU mode "A" or "S" is used, a separate security context is established on each data channel. The context is established by performing the GSSAPI handshake with the active-DTP calling gss\_init\_sec\_context() and the passive-DTP calling gss\_accept\_sec\_context(). No delegation need be done on these data channels.

Data channel protection via the PROT command MUST always be used in conjunction with the DCAU A or DCAU S commands. If a PROT level is set, then messages will be wrapped according to RFC 2228 Appendix I using the contexts established on each data channel. Tokens transferred over the data channels when either PROT or DCAU is used are not framed in any way when using GSI. (When implementing this specification with other GSSAPI mechanisms, a 4 byte, big endian, binary token length should precede all tokens).

If the DCAU mode or the PROT mode is changed between file transfers when caching data channels in extended block mode, all open data channels must be closed. This is because the GSI implementation does not support changing levels of protection on an existing connection.

## Index

- globus\_ftp\_control.c, 4
  - globus\_i\_ftp\_control\_debug\_level, 4
- globus\_ftp\_control.h
  - GLOBUS\_FTP\_CONTROL\_TCPBUFFER\_-  
AUTOMATIC, 11
  - GLOBUS\_FTP\_CONTROL\_TCPBUFFER\_-  
DEFAULT, 11
  - GLOBUS\_FTP\_CONTROL\_TCPBUFFER\_-  
FIXED, 11
- globus\_ftp\_control.h, 4
  - globus\_ftp\_control\_auth\_callback\_t, 9
  - globus\_ftp\_control\_auth\_info\_t, 8
  - globus\_ftp\_control\_auth\_requirements\_t, 9
  - globus\_ftp\_control\_callback\_t, 8
  - globus\_ftp\_control\_command\_callback\_t, 8
  - globus\_ftp\_control\_create\_data\_info, 12
  - globus\_ftp\_control\_data\_callback\_t, 9
  - globus\_ftp\_control\_data\_set\_interface, 11
  - globus\_ftp\_control\_data\_write\_stripe, 12
  - globus\_ftp\_control\_dcau\_mode\_e, 10
  - globus\_ftp\_control\_dcau\_mode\_t, 7
  - globus\_ftp\_control\_dcau\_subject\_t, 7
  - globus\_ftp\_control\_dcau\_t, 7
  - globus\_ftp\_control\_delay\_passive\_t, 11
  - globus\_ftp\_control\_layout\_t, 7
  - globus\_ftp\_control\_local\_layout, 11
  - globus\_ftp\_control\_mode\_e, 10
  - globus\_ftp\_control\_mode\_t, 7
  - GLOBUS\_FTP\_CONTROL\_MODULE, 6
  - globus\_ftp\_control\_parallelism\_mode\_e, 11
  - globus\_ftp\_control\_parallelism\_mode\_t, 7
  - globus\_ftp\_control\_parallelism\_t, 7
  - globus\_ftp\_control\_protection\_t, 11
  - globus\_ftp\_control\_release\_data\_info, 12
  - globus\_ftp\_control\_response\_callback\_t, 8
  - globus\_ftp\_control\_round\_robin\_t, 7
  - globus\_ftp\_control\_server\_callback\_t, 10
  - globus\_ftp\_control\_striping\_mode\_e, 11
  - globus\_ftp\_control\_striping\_mode\_t, 7
  - globus\_ftp\_control\_structure\_e, 11
  - globus\_ftp\_control\_structure\_t, 7
  - globus\_ftp\_control\_tcpbuffer\_automatic\_t, 8
  - globus\_ftp\_control\_tcpbuffer\_mode\_e, 11
  - globus\_ftp\_control\_tcpbuffer\_mode\_t, 8
  - globus\_ftp\_control\_tcpbuffer\_t, 8
  - globus\_ftp\_control\_type\_e, 10
  - globus\_ftp\_control\_type\_t, 7
  - globus\_X\_ftp\_control\_data\_write\_stripe, 13
- globus\_ftp\_control\_abort
  - globus\_ftp\_control\_client.c, 18
- globus\_ftp\_control\_auth\_callback\_t
  - globus\_ftp\_control.h, 9
- globus\_ftp\_control\_auth\_info\_compare
  - globus\_ftp\_control\_client.c, 20
- globus\_ftp\_control\_auth\_info\_init
  - globus\_ftp\_control\_client.c, 20
- globus\_ftp\_control\_auth\_info\_s, 2
- globus\_ftp\_control\_auth\_info\_t
  - globus\_ftp\_control.h, 8
- globus\_ftp\_control\_auth\_requirements\_t
  - globus\_ftp\_control.h, 9
- globus\_ftp\_control\_authenticate
  - globus\_ftp\_control\_client.c, 16
- globus\_ftp\_control\_callback\_t
  - globus\_ftp\_control.h, 8
- globus\_ftp\_control\_client.c, 14
  - globus\_ftp\_control\_abort, 18
  - globus\_ftp\_control\_auth\_info\_compare, 20
  - globus\_ftp\_control\_auth\_info\_init, 20
  - globus\_ftp\_control\_authenticate, 16
  - globus\_ftp\_control\_connect, 15
  - globus\_ftp\_control\_force\_close, 19
  - globus\_ftp\_control\_handle\_destroy, 14
  - globus\_ftp\_control\_handle\_init, 14
  - globus\_ftp\_control\_quit, 19
  - globus\_ftp\_control\_response\_copy, 16
  - globus\_ftp\_control\_response\_destroy, 16
  - globus\_ftp\_control\_send\_command, 17
- globus\_ftp\_control\_command\_callback\_t
  - globus\_ftp\_control.h, 8
- globus\_ftp\_control\_command\_copy
  - globus\_ftp\_control\_server.c, 32
- globus\_ftp\_control\_command\_destroy
  - globus\_ftp\_control\_server.c, 32
- globus\_ftp\_control\_command\_init
  - globus\_ftp\_control\_server.c, 31
- globus\_ftp\_control\_connect
  - globus\_ftp\_control\_client.c, 15
- globus\_ftp\_control\_create\_data\_info
  - globus\_ftp\_control.h, 12
  - globus\_ftp\_control\_data.c, 28
- globus\_ftp\_control\_data.c, 21
  - globus\_ftp\_control\_create\_data\_info, 28
  - globus\_ftp\_control\_data\_add\_channels, 23
  - globus\_ftp\_control\_data\_connect\_read, 22
  - globus\_ftp\_control\_data\_connect\_write, 23
  - globus\_ftp\_control\_data\_force\_close, 30
  - globus\_ftp\_control\_data\_get\_total\_data\_-  
channels, 24
  - globus\_ftp\_control\_data\_query\_channels, 24
  - globus\_ftp\_control\_data\_read, 27
  - globus\_ftp\_control\_data\_remove\_channels, 23
  - globus\_ftp\_control\_data\_send\_eof, 23
  - globus\_ftp\_control\_data\_set\_interface, 22
  - globus\_ftp\_control\_data\_write, 27
  - globus\_ftp\_control\_data\_write\_stripe, 29

- globus\_ftp\_control\_get\_pbsz, [27](#)
- globus\_ftp\_control\_local\_dcau, [26](#)
- globus\_ftp\_control\_local\_layout, [28](#)
- globus\_ftp\_control\_local\_mode, [26](#)
- globus\_ftp\_control\_local\_parallelism, [24](#)
- globus\_ftp\_control\_local\_pasv, [25](#)
- globus\_ftp\_control\_local\_pbsz, [26](#)
- globus\_ftp\_control\_local\_port, [25](#)
- globus\_ftp\_control\_local\_send\_eof, [24](#)
- globus\_ftp\_control\_local\_spas, [25](#)
- globus\_ftp\_control\_local\_spor, [25](#)
- globus\_ftp\_control\_local\_stru, [27](#)
- globus\_ftp\_control\_local\_tcp\_buffer, [26](#)
- globus\_ftp\_control\_local\_type, [25](#)
- globus\_ftp\_control\_release\_data\_info, [28](#)
- globus\_X\_ftp\_control\_data\_write\_stripe, [29](#)
- globus\_ftp\_control\_data\_add\_channels
  - globus\_ftp\_control\_data.c, [23](#)
- globus\_ftp\_control\_data\_callback\_t
  - globus\_ftp\_control.h, [9](#)
- globus\_ftp\_control\_data\_connect\_read
  - globus\_ftp\_control\_data.c, [22](#)
- globus\_ftp\_control\_data\_connect\_write
  - globus\_ftp\_control\_data.c, [23](#)
- globus\_ftp\_control\_data\_force\_close
  - globus\_ftp\_control\_data.c, [30](#)
- globus\_ftp\_control\_data\_get\_total\_data\_channels
  - globus\_ftp\_control\_data.c, [24](#)
- globus\_ftp\_control\_data\_query\_channels
  - globus\_ftp\_control\_data.c, [24](#)
- globus\_ftp\_control\_data\_read
  - globus\_ftp\_control\_data.c, [27](#)
- globus\_ftp\_control\_data\_remove\_channels
  - globus\_ftp\_control\_data.c, [23](#)
- globus\_ftp\_control\_data\_send\_eof
  - globus\_ftp\_control\_data.c, [23](#)
- globus\_ftp\_control\_data\_set\_interface
  - globus\_ftp\_control.h, [11](#)
  - globus\_ftp\_control\_data.c, [22](#)
- globus\_ftp\_control\_data\_write
  - globus\_ftp\_control\_data.c, [27](#)
- globus\_ftp\_control\_data\_write\_stripe
  - globus\_ftp\_control.h, [12](#)
  - globus\_ftp\_control\_data.c, [29](#)
- globus\_ftp\_control\_dcau\_mode\_e
  - globus\_ftp\_control.h, [10](#)
- globus\_ftp\_control\_dcau\_mode\_t
  - globus\_ftp\_control.h, [7](#)
- globus\_ftp\_control\_dcau\_subject\_s, [2](#)
- globus\_ftp\_control\_dcau\_subject\_t
  - globus\_ftp\_control.h, [7](#)
- globus\_ftp\_control\_dcau\_t
  - globus\_ftp\_control.h, [7](#)
- globus\_ftp\_control\_dcau\_u, [2](#)
- globus\_ftp\_control\_delay\_passive\_t
  - globus\_ftp\_control.h, [11](#)
- globus\_ftp\_control\_force\_close
  - globus\_ftp\_control\_client.c, [19](#)
- globus\_ftp\_control\_get\_pbsz
  - globus\_ftp\_control\_data.c, [27](#)
- globus\_ftp\_control\_handle\_destroy
  - globus\_ftp\_control\_client.c, [14](#)
- globus\_ftp\_control\_handle\_init
  - globus\_ftp\_control\_client.c, [14](#)
- globus\_ftp\_control\_layout.c, [30](#)
- globus\_ftp\_control\_layout\_t
  - globus\_ftp\_control.h, [7](#)
- globus\_ftp\_control\_layout\_u, [3](#)
- globus\_ftp\_control\_local\_dcau
  - globus\_ftp\_control\_data.c, [26](#)
- globus\_ftp\_control\_local\_layout
  - globus\_ftp\_control.h, [11](#)
  - globus\_ftp\_control\_data.c, [28](#)
- globus\_ftp\_control\_local\_mode
  - globus\_ftp\_control\_data.c, [26](#)
- globus\_ftp\_control\_local\_parallelism
  - globus\_ftp\_control\_data.c, [24](#)
- globus\_ftp\_control\_local\_pasv
  - globus\_ftp\_control\_data.c, [25](#)
- globus\_ftp\_control\_local\_pbsz
  - globus\_ftp\_control\_data.c, [26](#)
- globus\_ftp\_control\_local\_port
  - globus\_ftp\_control\_data.c, [25](#)
- globus\_ftp\_control\_local\_send\_eof
  - globus\_ftp\_control\_data.c, [24](#)
- globus\_ftp\_control\_local\_spas
  - globus\_ftp\_control\_data.c, [25](#)
- globus\_ftp\_control\_local\_spor
  - globus\_ftp\_control\_data.c, [25](#)
- globus\_ftp\_control\_local\_stru
  - globus\_ftp\_control\_data.c, [27](#)
- globus\_ftp\_control\_local\_tcp\_buffer
  - globus\_ftp\_control\_data.c, [26](#)
- globus\_ftp\_control\_local\_type
  - globus\_ftp\_control\_data.c, [25](#)
- globus\_ftp\_control\_mode\_e
  - globus\_ftp\_control.h, [10](#)
- globus\_ftp\_control\_mode\_t
  - globus\_ftp\_control.h, [7](#)
- GLOBUS\_FTP\_CONTROL\_MODULE
  - globus\_ftp\_control.h, [6](#)
- globus\_ftp\_control\_parallelism\_mode\_e
  - globus\_ftp\_control.h, [11](#)
- globus\_ftp\_control\_parallelism\_mode\_t
  - globus\_ftp\_control.h, [7](#)
- globus\_ftp\_control\_parallelism\_t
  - globus\_ftp\_control.h, [7](#)
- globus\_ftp\_control\_parallelism\_u, [3](#)
- globus\_ftp\_control\_protection\_t
  - globus\_ftp\_control.h, [11](#)
- globus\_ftp\_control\_quit
  - globus\_ftp\_control\_client.c, [19](#)

- globus\_ftp\_control\_read\_commands
  - globus\_ftp\_control\_server.c, [33](#)
- globus\_ftp\_control\_release\_data\_info
  - globus\_ftp\_control.h, [12](#)
  - globus\_ftp\_control\_data.c, [28](#)
- globus\_ftp\_control\_response\_callback\_t
  - globus\_ftp\_control.h, [8](#)
- globus\_ftp\_control\_response\_copy
  - globus\_ftp\_control\_client.c, [16](#)
- globus\_ftp\_control\_response\_destroy
  - globus\_ftp\_control\_client.c, [16](#)
- globus\_ftp\_control\_round\_robin\_s, [3](#)
- globus\_ftp\_control\_round\_robin\_t
  - globus\_ftp\_control.h, [7](#)
- globus\_ftp\_control\_send\_command
  - globus\_ftp\_control\_client.c, [17](#)
- globus\_ftp\_control\_send\_response
  - globus\_ftp\_control\_server.c, [34](#)
- globus\_ftp\_control\_server.c, [30](#)
  - globus\_ftp\_control\_command\_copy, [32](#)
  - globus\_ftp\_control\_command\_destroy, [32](#)
  - globus\_ftp\_control\_command\_init, [31](#)
  - globus\_ftp\_control\_read\_commands, [33](#)
  - globus\_ftp\_control\_send\_response, [34](#)
  - globus\_ftp\_control\_server\_accept, [32](#)
  - globus\_ftp\_control\_server\_authenticate, [33](#)
  - globus\_ftp\_control\_server\_handle\_destroy, [31](#)
  - globus\_ftp\_control\_server\_handle\_init, [31](#)
  - globus\_ftp\_control\_server\_listen, [31](#)
  - globus\_ftp\_control\_server\_stop, [32](#)
- globus\_ftp\_control\_server\_accept
  - globus\_ftp\_control\_server.c, [32](#)
- globus\_ftp\_control\_server\_authenticate
  - globus\_ftp\_control\_server.c, [33](#)
- globus\_ftp\_control\_server\_callback\_t
  - globus\_ftp\_control.h, [10](#)
- globus\_ftp\_control\_server\_handle\_destroy
  - globus\_ftp\_control\_server.c, [31](#)
- globus\_ftp\_control\_server\_handle\_init
  - globus\_ftp\_control\_server.c, [31](#)
- globus\_ftp\_control\_server\_listen
  - globus\_ftp\_control\_server.c, [31](#)
- globus\_ftp\_control\_server\_stop
  - globus\_ftp\_control\_server.c, [32](#)
- globus\_ftp\_control\_striping\_mode\_e
  - globus\_ftp\_control.h, [11](#)
- globus\_ftp\_control\_striping\_mode\_t
  - globus\_ftp\_control.h, [7](#)
- globus\_ftp\_control\_structure\_e
  - globus\_ftp\_control.h, [11](#)
- globus\_ftp\_control\_structure\_t
  - globus\_ftp\_control.h, [7](#)
- GLOBUS\_FTP\_CONTROL\_TCPBUFFER\_-  
AUTOMATIC
  - globus\_ftp\_control.h, [11](#)
- globus\_ftp\_control\_tcpbuffer\_automatic\_s, [3](#)

- globus\_ftp\_control\_tcpbuffer\_automatic\_t
  - globus\_ftp\_control.h, [8](#)
- GLOBUS\_FTP\_CONTROL\_TCPBUFFER\_-  
DEFAULT
  - globus\_ftp\_control.h, [11](#)
- globus\_ftp\_control\_tcpbuffer\_default\_t, [3](#)
- GLOBUS\_FTP\_CONTROL\_TCPBUFFER\_-  
FIXED
  - globus\_ftp\_control.h, [11](#)
- globus\_ftp\_control\_tcpbuffer\_fixed\_t, [3](#)
- globus\_ftp\_control\_tcpbuffer\_mode\_e
  - globus\_ftp\_control.h, [11](#)
- globus\_ftp\_control\_tcpbuffer\_mode\_t
  - globus\_ftp\_control.h, [8](#)
- globus\_ftp\_control\_tcpbuffer\_t, [4](#)
  - globus\_ftp\_control.h, [8](#)
- globus\_ftp\_control\_type\_e
  - globus\_ftp\_control.h, [10](#)
- globus\_ftp\_control\_type\_t
  - globus\_ftp\_control.h, [7](#)
- globus\_i\_ftp\_control\_debug\_level
  - globus\_ftp\_control.c, [4](#)
- globus\_X\_ftp\_control\_data\_write\_stripe
  - globus\_ftp\_control.h, [13](#)
  - globus\_ftp\_control\_data.c, [29](#)