

# globus xio Reference Manual

## 2.7

Generated by Doxygen 1.2.18

Fri Jun 26 15:42:26 2009

## Contents

<a href="#">1</a>	<a href="#">Globus XIO</a>	<a href="#">1</a>
<a href="#">2</a>	<a href="#">globus xio Module Index</a>	<a href="#">1</a>
<a href="#">3</a>	<a href="#">globus xio Data Structure Index</a>	<a href="#">3</a>
<a href="#">4</a>	<a href="#">globus xio File Index</a>	<a href="#">3</a>
<a href="#">5</a>	<a href="#">globus xio Page Index</a>	<a href="#">4</a>
<a href="#">6</a>	<a href="#">globus xio Module Documentation</a>	<a href="#">4</a>
<a href="#">7</a>	<a href="#">globus xio Data Structure Documentation</a>	<a href="#">70</a>
<a href="#">8</a>	<a href="#">globus xio File Documentation</a>	<a href="#">71</a>
<a href="#">9</a>	<a href="#">globus xio Page Documentation</a>	<a href="#">78</a>

## 1 Globus XIO

The Globus eXtensible Input Output library.

[The globusxio user API.](#)  
[User API Assistance.](#)  
[Globus XIO Driver](#)  
[Driver Programming](#)

## 2 globus xio Module Index

### 2.1 globus xio Modules

Here is a list of all modules:

<a href="#">The globus.xio user API.</a>	<a href="#">4</a>
<a href="#">User API Assistance.</a>	<a href="#">13</a>
<a href="#">Globus XIO Driver</a>	<a href="#">15</a>
<a href="#">Driver Programming</a>	<a href="#">17</a>
<a href="#">Driver Programming: String options</a>	<a href="#">26</a>
<a href="#">Globus XIO File Driver</a>	<a href="#">27</a>
<a href="#">Opening/Closing</a>	<a href="#">28</a>

Reading/Writing	28
Env Variables	28
Attributes and Cntls	29
Types	33
Error Types	35
Globus XIO HTTP Driver	35
Opening/Closing	36
Reading/Writing	36
Server	36
Attributes and Cntls	36
Error Types	41
Globus XIO MODE _E Driver	41
Opening/Closing	42
Reading/Writing	42
Server	42
Env Variables	42
Attributes and Cntls	42
Types	46
Error Types	46
Globus XIO ORDERING Driver	46
Opening/Closing	47
Reading/Writing	47
Env Variables	47
Attributes and Cntls	47
Types	49
Error Types	49
Globus XIO TCP Driver	50
Opening/Closing	50
Reading/Writing	50

Server	51
Env Variables	51
Attributes and Cntls	51
Types	60
Error Types	61
Globus XIO UDP Driver	61
Opening/Closing	62
Reading/Writing	62
Env Variables	62
Attributes and Cntls	63
Types	69
Error Types	69

## 3 globus xio Data Structure Index

### 3.1 globus xio Data Structures

Here are the data structures with brief descriptions:

<a href="#">globus.xio.http_header.t</a> (HTTP Header)	70
--	----

## 4 globus xio File Index

### 4.1 globus xio File List

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

<a href="#">globus.xio_file_driver.h</a> (Header file for XIO File Driver)	71
<a href="#">globus.xio.http.h</a> (Globus XIO HTTP Driver Header)	72
<a href="#">globus.xio_mode_e_driver.h</a> (Header file for XIO MODE_E Driver)	74
<a href="#">globus.xio_ordering_driver.h</a> (Header file for XIO ORDERING Driver)	75
<a href="#">globus.xio.tcp_driver.h</a> (Header file for XIO TCP Driver)	75
<a href="#">globus.xio_udp_driver.h</a> (Header file for XIO UDP Driver)	77

## 5 globus xio Page Index

### 5.1 globus xio Related Pages

Here is a list of all related documentation pages:

Data descriptors	78
Todo List	79

## 6 globus xio Module Documentation

### 6.1 The globusxio user API.

#### Typedefs

```
typedef void( globusxio\_acceptcallback )(globusxio_servlet server, globusxio_handle_t handle, globus_result_t result, void userarg)
typedef void( globusxio\_servercallback )(globusxio_servlet server, void userarg)
typedef globus_bool_t( globusxio\_timeoutcallback )(globusxio_handle_t handle, globusxio\_operation\_type\_t type, void userarg)
typedef void( globusxio\_callback )(globusxio_handle_t handle, globus_result_t result, void userarg)
typedef void( globusxio\_datacallback )(globusxio_handle_t handle, globus_result_t result, globus_byte_t buffer, globus_size_t len, globus_size_t nbytes, globusxio_data_descriptor_t datadesc, void userarg)
typedef void( globusxio\_iovec\_callback )(globusxio_handle_t handle, globus_result_t result, globusxio_iovec_t iovec, int count, globus_size_t nbytes, globusxio_data_descriptor_t datadesc, void userarg)
typedef enum globus\_xio\_op\_type\_e globusxio\_operation\_type\_t
```

#### Enumerations

```
enum globus\_xio\_op\_type\_e
enum globusxio\_handlecmd\_t { GLOBUS\_XIO\_GET\_LOCAL\_CONTACT = 12345, GLOBUS\_XIO\_GET\_LOCAL\_NUMERIC\_CONTACT, GLOBUS\_XIO\_GET\_REMOTE\_CONTACT, GLOBUS\_XIO\_GET\_REMOTE\_NUMERIC\_CONTACT, GLOBUS\_XIO\_SEEK, GLOBUS\_XIO\_SET\_STRING\_OPTIONSg
```

#### Functions

```
globus_result_t globusxio\_attr\_init (globusxio_attr_t attr)
globus_result_t globusxio\_attr\_cntl (globusxio_attr_t attr, globusxio_driver_t driver, int cmd,...)
globus_result_t globusxio\_attr\_copy (globusxio_attr_t dst, globusxio_attr_t src)
globus_result_t globusxio\_attr\_destroy (globusxio_attr_t attr)
globus_result_t globusxio\_stackinit (globusxio_stack_t stack, globusxio_attr_t stackattr)
globus_result_t globusxio\_stackpushdriver (globusxio_stack_t stack, globusxio_driver_t driver)
globus_result_t globusxio\_stackcopy (globusxio_stack_t dst, globusxio_stack_t src)
globus_result_t globusxio\_stackdestroy (globusxio_stack_t stack)
globus_result_t globusxio\_servercreate (globusxio_servlet server, globusxio_attr_t serverattr, globusxio_stack_t stack)
globus_result_t globusxio\_servergetcontactstring (globusxio_servlet server, char contactstring)
```

```

globusresultt globusxio_serverregisterclose (globusxio_servert server, globusxio_servercallback cb,
void userarg)
globusresultt globusxio_serverclose(globusxio_servert server)
globusresultt globusxio_servercntl (globusxio_servert server, globusxio_driver_t driver, int cmd,...)
globusresultt globusxio_serveraccept(globusxio_handlet out.handle, globusxio_servert server)
globusresultt globusxio_serverregisteraccept(globusxio_servert server, globusxio_acceptcallback cb,
void userarg)
globusresultt globusxio_handlecreate(globusxio_handlet handle, globusxio_stack_t stack)
globusresultt globusxio_datadescriptorinit (globusxio_datadescriptor_t datadesc, globusxio_handle-
t handle)
globusresultt globusxio_datadescriptordestroy(globusxio_datadescriptor_t datadesc)
globusresultt globusxio_datadescriptorcntl (globusxio_datadescriptor_t datadesc, globusxio_driver-
t driver, int cmd,...)
globusresultt globusxio_handlecntl (globusxio_handlet handle, globusxio_driver_t driver, int cmd,...)
globusresultt globusxio_registeropen(globusxio_handlet handle, const charcontactstring, globusxio_-
attr_t attr, globusxio_callback cb, void userarg)
globusresultt globusxio_open(globusxio_handlet handle, const charcontactstring, globusxio_attr_t attr)
globusresultt globusxio_registerread (globusxio_handlet handle, globusbyte_t buffer, globussize-
t buffer_length, globussize_t waitforbytes, globusxio_datadescriptor_t datadesc, globusxio_datacallbackt
cb, void userarg)
globusresultt globusxio_read(globusxio_handlet handle, globusbyte_t buffer, globussize_t buffer_length,
globussize_t waitforbytes, globussize_t nbytes, globusxio_datadescriptor_t datadesc)
globusresultt globusxio_registerreadv (globusxio_handlet handle, globusxio_iovec_t iovec, int iovec-
count, globussize_t waitforbytes, globusxio_datadescriptor_t datadesc, globusxio_iovec.callbackt cb, void
userarg)
globusresultt globusxio_readv (globusxio_handlet handle, globusxio_iovec_t iovec, int ioveccount,
globussize_t waitforbytes, globussize_t nbytes, globusxio_datadescriptor_t datadesc)
globusresultt globusxio_registerwrite (globusxio_handlet handle, globusbyte_t buffer, globussize-
t buffer_length, globussize_t waitforbytes, globusxio_datadescriptor_t datadesc, globusxio_datacallbackt
cb, void userarg)
globusresultt globusxio_write (globusxio_handlet handle, globusbyte_t buffer, globussize_t buffer_length,
globussize_t waitforbytes, globussize_t nbytes, globusxio_datadescriptor_t datadesc)
globusresultt globusxio_registerwritev (globusxio_handlet handle, globusxio_iovec_t iovec, int iovec-
count, globussize_t waitforbytes, globusxio_datadescriptor_t datadesc, globusxio_iovec.callbackt cb, void
userarg)
globusresultt globusxio_writev (globusxio_handlet handle, globusxio_iovec_t iovec, int ioveccount,
globussize_t waitforbytes, globussize_t nbytes, globusxio_datadescriptor_t datadesc)
globusresultt globusxio_registerclose (globusxio_handlet handle, globusxio_attr_t attr, globusxio_-
callbackt cb, void userarg)
globusresultt globusxio_close(globusxio_handlet handle, globusxio_attr_t attr)
globusresultt globusxio_handlecreatefrom_url (globusxio_handlet out.h, const char scheme, globus
xio_attr_t attr, char paramstring)
EXTERN_C_END globusresultt globusxio_handlecntl (handle, driver, GLOBUSXIO_GET_LOCAL_-
CONTACT, char contactstring.out)
globusresultt globusxio_handlecntl (handle, driver, GLOBUSXIO_SEEK, globusoff_t offset)
globusresultt globusxio_handlecntl (handle, driver, GLOBUSXIO_SET_STRING_OPTIONS, char
con g_string)

```

### 6.1.1 Typedef Documentation

6.1.1.1 `typedef void( globus_xio_acceptcallback_t)( globus_xio_server_t server, globus_xio_handle_t handle, globus_result_t result, void user_arg)`

Callback signature for accept.

When a registered accept operation completes the users function of this signature is called.

Parameters:

server The server object on which the accept was registered.

handle The newly created handle that was created by the accept operation.

result A result code indicating the success of the accept operation. `GLOBUS_SUCCESS` indicates a successful accept.

user\_arg A user argument that is threaded from the registration to the callback.

6.1.1.2 `typedef void( globus_xio_server_callback_t)( globus_xio_server_t server, void user_arg)`

Server callback signature.

This is the generic server callback signature. It is currently only used for the register close operation.

6.1.1.3 `typedef globus_bool_t( globus_xio_timeout_callback_t)( globus_xio_handle_t handle, globus_xio_operation_type_t type, void user_arg)`

The timeout callback function signature.

Parameters:

handle The handle the handle on which the timeout operation was requested.

type The type of operation that timed out: `GLOBUSXIO_OPERATION_OPEN` `GLOBUSXIO_OPERATION_CLOSE` `GLOBUSXIO_OPERATION_READ` `GLOBUSXIO_OPERATION_WRITE`

arg A user arg threaded throw to the callback.

6.1.1.4 `typedef void( globus_xio_callback_t)( globus_xio_handle_t handle, globus_result_t result, void user_arg)`

`globus_xio_callback_t`

This callback is used for the open and close asynchronous operations.

6.1.1.5 `typedef void( globus_xio_data_callback_t)( globus_xio_handle_t handle, globus_result_t result, globus_byte_t buffer, globus_size_t len, globus_size_t nbytes, globus_xio_data_descriptor_t data_desc, void user_arg)`

`globus_xio_data_callback_t`

This callback is used for asynchronous operations that send or receive data.

on eof, result will be of type `GLOBUSXIO_ERROR_EOF`

6.1.1.6 `typedef void( globus_xio_ivec.callback_t)( globus_xio_handle_t handle, globus_result_t result, globus_xio_ivec_t ivec, int count, globussize_t nbytes, globusxio_data_descriptor_t data_desc, void user_arg)`

`globus_xio_ivec.callback_t`

This callback is used for asynchronous operations that send or receive data with an ivec structure.

on eof, result will be of type `GLOBUSXIO_ERROR_EOF`

6.1.1.7 `typedef enum globus_xio_op_type_e globus_xio_operation_type_t`

Operation types.

An enumeration of operation types. Used in the timeout callback to indicate what operation typed timedout.

## 6.1.2 Enumeration Type Documentation

6.1.2.1 `enum globus_xio_op_type_e`

Operation types.

An enumeration of operation types. Used in the timeout callback to indicate what operation typed timedout.

6.1.2.2 `enum globus_xio_handle_cmd_t`

Common driver handle cntls.

Enumeration values:

`GLOBUS_XIO_GET_LOCAL_CONTACT` See usage for [globus\\_xio\\_handlectl](#).

`GLOBUS_XIO_GET_LOCAL_NUMERIC_CONTACT` See usage for [globus\\_xio\\_handlectl](#).

`GLOBUS_XIO_GET_REMOTE_CONTACT` See usage for [globus\\_xio\\_handlectl](#).

`GLOBUS_XIO_GET_REMOTE_NUMERIC_CONTACT` See usage for [globus\\_xio\\_handlectl](#).

`GLOBUS_XIO_SEEK` See usage for [globus\\_xio\\_handlectl](#).

`GLOBUS_XIO_SET_STRING_OPTIONS` See usage for [globus\\_xio\\_handlectl](#).

## 6.1.3 Function Documentation

6.1.3.1 `globus_result_t globus_xio_attr_init( globus_xio_attr_t attr)`

Intialize a globus xio attribute.

Parameters:

`attr` upon return from this function this out parameter will be initialized. Once the user is nished with the attribute they should make sure they destroy it in order to free resources associated with it.

6.1.3.2 `globus_result_t globus_xio_attr_ctl( globus_xio_attr_t attr, globus_xio_driver_t driver, int cmd, ...)`

Manipulate the values associated in the attr.

This function provides a means to access the attr structure. What exactly this function does is determined by the value in the parameter `cmd` and the value of the parameter `driver`. When the driver parameter is `NULL` it indicates that this function applies to general globus xio values. If it is not `NULL` it indicates that the function will effect driver specific values. Each driver is responsible for defining its own enumeration of values for `cmd` and the var args associated with that command.



## Parameters:

`attr` the attribute structure to be manipulated.

`driver` This parameter indicates which driver the user would like to perform the requested operation. If this parameter is NULL this request will be scoped to general attribute functions.

`cmd` an enum that determines what specific operation the user is requesting. Each driver will determine the value for this enumeration.

6.1.3.3 `globusresult_t globus_xio_attr_copy (globus_xio_attr_t dst, globus_xio_attr_t src)`

Copy an attribute structure.

6.1.3.4 `globusresult_t globus_xio_attr_destroy (globus_xio_attr_t attr)`

Clean up resources associated with an attribute.

## Parameters:

`attr` Upon completion of this function all resources associated with this structure will be returned to the system and the `attr` will no longer be valid.

6.1.3.5 `globusresult_t globus_xio_stack_init (globus_xio_stack_t stack, globus_xio_attr_t stackattr)`

Initialize a stack object.

6.1.3.6 `globusresult_t globus_xio_stack_push_driver (globus_xio_stack_t stack, globus_xio_driver_t driver)`

Push a driver onto a stack.

No attrs are associated with a driver. The stack represents the ordered lists of transform drivers and 1 transport driver. The transport driver must be pushed on first.

6.1.3.7 `globusresult_t globus_xio_stack_copy (globus_xio_stack_t dst, globus_xio_stack_t src)`

Copy a stack object.

6.1.3.8 `globusresult_t globus_xio_stack_destroy (globus_xio_stack_t stack)`

Destroy a stack object.

6.1.3.9 `globusresult_t globus_xio_server_create (globus_xio_server_t server, globus_xio_attr_t serverattr, globus_xio_stack_t stack)`

Create a server object.

This function allows the user to create a server object which can then be used to accept connections.

## Parameters:

`server` An out parameter. Once the function successfully returns this will point to a valid server object.

`serverattr` an attribute structure used to alter the default server initialization. This will mostly be used in a driver specific manner. can be NULL.

`stack`

6.1.3.10 `globus_result_t globus_xio_server_get_contact_string (globus_xio_server_t server, char *contact_string)`

get contact string

This function allows the user to get the contact string for a server. this string could be used as the contact string for the client side.

Parameters:

server An initialized server handle created with [globus\\_xio\\_server\\_create\(\)](#)

contactstring an out varibale. Will point to a newly allocated string on success. must be freed by the caller.

6.1.3.11 `globus_result_t globus_xio_server_register_close (globus_xio_server_t server, globus\_xio\_server\_callback\_t cb, void *user_arg)`

post a close on a server object

This function registers a close operation on a server. When the user function pointed to by parameter cb is called the server object is closed.

6.1.3.12 `globus_result_t globus_xio_server_close (globus_xio_server_t server)`

A blocking server close.

6.1.3.13 `globus_result_t globus_xio_server_cntl (globus_xio_server_t server, globus_xio_driver_t driver, int cmd, ...)`

Touch driver specific information in a server object.

This function allows the user to communicate directly with a driver in association with a server object. The driver defines what operations can be performed.

6.1.3.14 `globus_result_t globus_xio_server_accept (globus_xio_handle_t *out_handle, globus_xio_server_t server)`

Accept a connection.

This function will accept a connection on the given server object and the parameter `out_handle` will be valid if the function returns successfully.

6.1.3.15 `globus_result_t globus_xio_server_register_accept (globus_xio_server_t server, globus\_xio\_accept\_callback\_t cb, void *user_arg)`

Asynchronous accept.

This function posts a nonblocking accept. Once the operation has completed the user function pointed to by the parameter cb is called.

6.1.3.16 `globus_result_t globus_xio_handle_create (globus_xio_handle_t *handle, globus_xio_stack_t stack)`

Initialize a handle for client opens.

This function will initialize a handle for active opens (client side connections).

6.1.3.17 `globusresult_t globus_xio_data_descriptor_init (globus_xio_data_descriptor_t data_desc, globus_xio_handle_t handle)`

Initialize a data descriptor.

Parameters:

`data_desc` An out parameter. The data descriptor to be initialized.

`handle` The handle this data descriptor will be used with. This parameter is required in order to optimize the code handling the data descriptors use.

6.1.3.18 `globusresult_t globus_xio_data_descriptor_destroy (globus_xio_data_descriptor_t data_desc)`

clean up a data descriptor.

6.1.3.19 `globusresult_t globus_xio_data_descriptor_cntl (globus_xio_data_descriptor_t data_desc, globus_xio_driver_t driver, int cmd, ...)`

Touch driver specific data in data descriptors.

This function allows the user to communicate directly with a driver in association with a data descriptor. The driver defines what operations can be performed.

6.1.3.20 `globusresult_t globus_xio_handle_cntl (globus_xio_handle_t handle, globus_xio_driver_t driver, int cmd, ...)`

Touch driver specific information in a handle object.

This function allows the user to communicate directly with a driver in association with a handle object. The driver defines what operations can be performed.

pass the driver to control a specific driver pass NULL for driver for XIO specific cntls pass `GLOBUS_XIO_QUERY` for driver to try each driver in order until success

6.1.3.21 `globusresult_t globus_xio_register_open (globus_xio_handle_t handle, const char *contactstring, globus_xio_attr_t attr, globus\_xio\_callback\_t cb, void *user_arg)`

Open a handle.

Creates an open handle based on the state contained in the given stack.

No operation can be performed on a handle until it is initialized and then opened. If an already open handle used the information contained in that handle will be destroyed.

Parameters:

`handle` The handle created with [globus\\_xio\\_handlecreate\(\)](#) or [globus\\_xio\\_server\\_register\\_accept\(\)](#) that is to be opened.

`attr` how to open attribute. can be NULL

`cb` The function to be called when the open operation completes.

`user_arg` A user pointer that will be threaded through to the callback.

`contactstring` An url describing the resource. NULL is allowed. Drivers interpret the various parts of this url as described in their documentation. An alternative form is also supported: if `contactstring` does not specify a scheme (e.g. `http://` ) and it contains a ':', it will be parsed as a host:port pair. if it does not contain a ':', it will be parsed as the path

the following are examples of valid formats:

```

<path to file>
host-name ":" <service or port>
"file:" <path to file>
<scheme> "://" [ "/" [ <path to resource> ] ]
<scheme> "://" location [ "/" [ <path to resource> ] ]
location:
    [ auth-part ] host-part
auth-part:
    <user> [ ":" <password> ] "@"
host-part:
    [ "<" <subject> ">" ] host-name [ ":" <port or service> ]
host-name:
    <hostname> | <dotted quad> | "[" <ipv6 address> "]"

```

Except for use as the above delimiters, the following special characters MUST be encoded with the %HH format where H == hex char.

```

"/" and "@" in location except subject
"<" and ">" in location
":" everywhere except ipv6 address and subject
%" everywhere (can be encoded with %HH or %%)

```

6.1.3.22 `globus_result_t globus_xio_open (globus_xio_handle_t handle, const char contactstring, globus_xio_attr_t attr)`

blocking open

6.1.3.23 `globus_result_t globus_xio_register_read (globus_xio_handle_t handle, globus_byte_t buffer, globus_size_t buffer_length, globus_size_t waitforbytes, globus_xio_data_descriptor_t data_desc, globus\_xio\_data\_callback\_t cb, void user_arg)`

Read data from a handle.

6.1.3.24 `globus_result_t globus_xio_read (globus_xio_handle_t handle, globus_byte_t buffer, globus_size_t buffer_length, globus_size_t waitforbytes, globus_size_t nbytes, globus_xio_data_descriptor_t data_desc)`

Read data from a handle.

6.1.3.25 `globus_result_t globus_xio_register_readv (globus_xio_handle_t handle, globus_xio_iovec_t iovec, int iovec_count, globus_size_t waitforbytes, globus_xio_data_descriptor_t data_desc, globus\_xio\_iovec\_callback\_t cb, void user_arg)`

Read data from a handle into a `globus_xio_iovec_t` (struct iovec).

6.1.3.26 `globus_result_t globus_xio_readv (globus_xio_handle_t handle, globus_xio_iovec_t iovec, int iovec_count, globus_size_t waitforbytes, globus_size_t nbytes, globus_xio_data_descriptor_t data_desc)`

Read data from a handle into a `globus_xio_iovec_t` (struct iovec).

6.1.3.27 `globus_result_t globus_xio_register_write (globus_xio_handle_t handle, globus_byte_t buffer, globus_size_t buffer_length, globus_size_t waitforbytes, globus_xio_data_descriptor_t data_desc, globus\_xio\_data\_callback\_t cb, void user_arg)`

Write data to a handle.

6.1.3.28 `globus_result_t globus_xio_write (globus_xio_handle_t handle, globus_byte_t buffer, globus_size_t buffer_length, globus_size_t waitforbytes, globus_size_t nbytes, globus_xio_data_descriptor_t data_desc)`

Write data to a handle.

6.1.3.29 `globus_result_t globus_xio_register_writew (globus_xio_handle_t handle, globus_xio_iovec_t iovec, int iovec_count, globus_size_t waitforbytes, globus_xio_data_descriptor_t data_desc, globus\_xio\_iovec\_callback\_t cb, void user_arg)`

Write data to a handle from a `globus_xio_iovec_t` (struct iovec).

6.1.3.30 `globus_result_t globus_xio_writew (globus_xio_handle_t handle, globus_xio_iovec_t iovec, int iovec_count, globus_size_t waitforbytes, globus_size_t nbytes, globus_xio_data_descriptor_t data_desc)`

Write data to a handle from a `globus_xio_iovec_t` (struct iovec).

6.1.3.31 `globus_result_t globus_xio_register_close (globus_xio_handle_t handle, globus_xio_attr_t attr, globus\_xio\_callback\_t cb, void user_arg)`

Close a handle.

This functions servers as a destroy for the handle. As soon as the operations completes (the callback is called). The handle is destroyed.

Parameters:

handle the handle to be closed.

attr how to close attribute

cb The function to be called when the close operation completes.

user\_arg A user pointer that will be threaded through to the callback.

6.1.3.32 `globus_result_t globus_xio_close (globus_xio_handle_t handle, globus_xio_attr_t attr)`

Blocking close.

6.1.3.33 `globus_result_t globus_xio_handle_create_from_url (globus_xio_handle_t out_h, const char scheme, globus_xio_attr_t attr, char param_string)`

Initializes a handle based on the scheme given.

Parameters:

out\_h An uninitialized handle that will be initialized in the function to correspond to the scheme given. This handle should be used for any I/O operations.

scheme A string containing the protocol which the handle should be initialized to. The string can either be a protocol by itself, for example, "http", or a complete scheme such as <http://www.example.com>.

attr Attribute to be used for setting parameter string. It is initialized by the function. Can be NULL if attributes are not being used.

`param.string` A string containing attributes to be set for the drivers associated with the scheme. This should be in the form "protocol1:option1=value1;option2=value2,protocol2:option1=value1; option2=value2" Can be NULL if attributes are not being used.

6.1.3.34 `globus_result_t globus_xio_handle_ctl (handle, driver, GLOBUS_XIO_GET_LOCAL_CONTACT, char contactstring_out)`

Get local connection info.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`contactstring_out` A pointer to a contact string for the local end of a connected handle. Where possible, it will be in symbolic form (FQDN).

The user must free the returned string.

See also:

[globus\\_xio\\_server\\_get\\_contactstring\(\)](#)

6.1.3.35 `globus_result_t globus_xio_handle_ctl (handle, driver, GLOBUS_XIO_SEEK, globus_off_t offset)`

Reposition read/write offset.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`offset` Specify the desired offset.

6.1.3.36 `globus_result_t globus_xio_handle_ctl (handle, driver, GLOBUS_XIO_SET_STRING_OPTIONS, char con_g_string)`

Set the driver specific configuration string.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

The format of the string is defined by the driver. It is typically a set of key=value pairs

Parameters:

`con_g_string` The driver specific parameter string.

## 6.2 User API Assistance.

Help understanding the ~~globus~~ xio api.

Stack Constuction.

The driver stack that is used for a given xio handle is constructed using a ~~globus~~ stack. Each driver is loaded by name and pushed onto a stack.

stack setup example:

```
// First load the drivers
globus_xio_driver_load("tcp", &tcp_driver);
globus_xio_driver_load("gsi", &gsi_driver);

//build the stack
globus_xio_stack_init(&stack);
globus_xio_stack_push_driver(stack, tcp_driver, NULL);
globus_xio_stack_push_driver(stack, gsi_driver, NULL);
```

## Servers

A server data structure provides functionality for passive opens. A server is initialized and bound to a protocol stack and set of attributes with the function `globus_xio_servercreate()`. Once a server is created many "connections" can be accepted. Each connection will result in an initialized handle which can later be opened.

```
globus_xio_server_t      server;
globus_xio_attr_t        attr;

globus_xio_attr_init(&attr);
globus_xio_server_create(&server_handle, attr, stack);
globus_xio_server_accept(&handle, server);
```

## Handle Construction

There are two ways to create a handle. The first is for use as a client (one that is doing an active open). The function: `globus_xio_handlecreate()` is used to create such a handle and bind that handle to a protocol stack.

```
globus_xio_handle_create(&handle, stack);
```

The second means of creating a handle is for use as a server (one that is doing a passive open). This is created by accepting a connection on a server handle with the function `globus_xio_serveraccept()` or `globus_xio_serverregister-accept()`.

Mutable attrs can be altered via a call to `globus_xio_handlecntl()` described later.

```
globus_xio_server_accept(&xio_handle, server_handle);
```

once a handle is initialized the user can call `globus_xio_open()` to begin the open process.

## Timeouts

A user can set a timeout value for any io operation. Each IO operation (open close read write) can have its own timeout value. If no timeout is set the operation will be allowed to infinitely block.

When time expires the outstanding operation is canceled. If the timeout callback for the given operation is not NULL it is called first to notify the user that the operation timed out and give the user a chance to ignore that timeout. If canceled, the user will get the callback they registered for the operation as well, but it will come with an error indicating that it has been canceled.

It is possible that part of an io operation will complete before the timeout expires. In this case the operation can still be canceled. The user will receive their IO callback with an error set and the length value appropriately set to indicate how much of the operation completed.

## Data Descriptor

The data descriptor ADT gives the user a means of attaching/extracting meta data to a read or write operation. Things like offset, out of band message, and other driver specific meta data are contained in the data descriptor. Data descriptors are passed to `globus_xio` in `globus_xio_read()` and `globus_xio_write()`. Within the `globus_xio` framework it is acceptable to pass NULL instead of a valid data descriptor,

```
ex:
globus_xio_data_descriptor_init(&desc);
globus_xio_data_descriptor_cntl(desc,
    tcp_driver,
    GLOBUS_XIO_TCP_SET_SEND_FLAGS,
    GLOBUS_XIO_TCP_SEND_OOB);
```

### User Attributes

Globus XIO uses a single attribute object for all of its functions. Attributes give the user an extensible mechanism to alter default values which control parameters in an operation.

In most of the globus xio user api functions a user passes an attribute as a parameter. In many cases the user may ignore the attribute parameter and just pass in NULL. However at times the user will wish to tweak the operation.

The attribute structure is used for this tweaking.

There are only three attribute functions: [The globusxio user API.Attributes and Cntls](#) and [The globusxio user API..](#) The init and destroy functions are very simple and require little explanation. Before an attribute can be used it must be initialized, and to clean up all memory associated with it the user must call destroy on it.

The function [Attributes and Cntls](#) manipulates values in the attribute. For more info on it [See Attributes and Cntls](#)

## 6.3 Globus XIO Driver

Globus XIO introduces a notion of a driver stack to its API. Within globusxio every IO operation must occur on a globusxio handle. Associated with each handle is a stack of drivers. A driver is a module piece of code that implements the globusxio driver interface. The purpose of a driver is manipulate data passed in by the user in some way. Each driver in a stack will serve its own unique purpose.

IO operations pass from driver to driver, starting at the top of the stack and ending at the bottom. When the bottom layer driver finishes with the operation it signals globusxio that it has completed. Completion notification then follows the driver stack up to the top.

### Driver Types:

#### Transport driver:

A transport driver is one that is responsible for actually putting bytes onto the wire. For example: A TCP driver or a UDP driver would be an example of transport drivers.

Per driver stack there must be exactly one transport driver and must be at the bottom of the stack. A transform driver is defined by its lack of passing an operation to the next driver in the stack. This type of driver does not rely on globusxio for further completion of an operation, rather it is self sufficient in this task.

#### Transform driver:

A transform driver is any intermediate driver in the stack. These drivers are identified by their reliance on the driver stack to complete the operation. These drivers must pass the operation down the stack because they cannot complete it themselves. An example of a transform driver would be a gsi driver. This driver would wrap and unwrap messages, but would not be able to complete the transport itself, so it would rely on the remaining drivers in the stack.

### Driver API

The globus xio driver api is a set of functions and interfaces to allow a developer to create a backend driver for globusxio. To create a driver the user must implement all of the interface functions in the driver specification. There are also a set of functions provided to assist the driver author in implementation.

#### Quick Start:

Four basic driver needs the user will have to pay attention to a few new structures and concepts.

```
globus_xio_operation_t
```

This structure represents a request for an operation. If the driver can service the operation it does so and then calls the appropriate `finishoperation()` function. If the driver cannot completely service the operation it can pass() it along to the next driver in the stack. As soon as the operation structure is either finished or passed it is no longer valid for use in any other function.



`globus_xio_driver_handle_t`

A driver handle represents an open handle to the driver stack for xio. The driver obtains a handle by calling `globusxio_driver_open()`. When the open operation completes (its callback is called) the driver then has a driver handle. The driver handle allows the user to do some complex things that will be described later.

`globus_xio_stack_t`

This structure provides the driver with information about the driver stack. It is mainly used for creating a driver handle as a parameter to `globusxio_driver_open()`.

#### Typical Sequence:

Here is a typical sequence of events for a globus transform driver:

##### Open

`globusxio_driver_open` is called. The user calls `globusxio_driver_open()` passing it the operation and the stack and a callback. When the open callback is called the driver is given a new operation as a parameter. The driver will then call `globusxio_driver_nished_open()` passing it the now initialized driver handle and the newly received operation. The call to `globusxio_driver_nished_open()` does two things: 1) it tells globusio that this driver has nished its open operation, and 2) it gives xio the driver handle (which contains information on the drivers below it).

##### Read/Write

The read or write interface function is called. It receives an operation as a parameter. The driver then calls the appropriate pass operation and waits for the callback. When the callback is received the driver calls `globusxio_driver_nished` passing in the operation structure it received in the callback.

##### Close

The close interface function is called and is passed an operation and a driver handle. The driver will call `globusxio_driver_close()` passing it the operation. When the close callback is received the driver calls `globusxio_driver_nished_close()` passing it the operation received in the close callback and the driver handle received in the interface function. At this point the driver handle is no longer valid.

#### Advanced Driver Programming

The typical driver implementation is described above. However globus allows driver authors to do more advanced things. Some of these things will be explored here.

##### Read Ahead

Once a driver handle is open a driver can spawn operation structures from it. This gives the driver the ability to request io from the driver stack before it receives a call to its own interface io interface function. So if a driver wishes to read ahead it does the following:

it creates an operation by calling `globusxio_driver_createoperation()` and passing it the driver handle it is interesting in using.

call `globusxio_driver_read()` using this operation. When the read callback is received the driver may call `globusxio_driver_nished_operation()` on the op it receives (this ultimately results in very little, since this operation was started by this driver, but it is good practice and will free up resources that would otherwise leak).

Now when the user finally does receive a read interface call from globus it can immediately nish it using the operation it just received as a parameter and updating the `iovec` structure to represent the read that already occurred.

##### Preopening handles.

Once the driver has received a `globusxio_driver_stack_t` it can open a driver handle. The `globusxio_driver_stack_t` comes in the call to the interface function `globusxio_server/clientinit_t()`. The driver uses this structure in a call to `globusxio_driver_open()`. When this functionality completes the driver has an initialized driver handle and can use it to create operations as described above. The driver can now hang onto the handle until it receives an open interface function call. At which time it can call `globusxio_driver_nished_open()` passing in the driver handle and thereby glueing the pre opened driver handle with the requested globus operation.

## 6.4 Driver Programming

The set of interface functions that the driver author must implement to create a driver and the functions to assist in the creation.

### Typedefs

```
typedef void( globusxio_driver_callbackt )(globusxio_operationnt op, globusresultt result, void userarg)
typedef void( globusxio_driver_datacallbackt )(globusxio_operationnt op, globusresultt result, globus-
size_t nbytes, void userarg)
typedef globusresultt( globusxio_driver_attr_init_t )(void out_driver_attr)
typedef globusresultt( globusxio_driver_attr_copy_t )(void dst, void src)
typedef globusresultt( globusxio_driver_attr_destroy_t )(void driver_attr)
typedef globusresultt( globusxio_driver_attr_cntl_t )(void attr, int cmd, valist ap)
typedef globusresultt( globusxio_driver_serverinit_t )(void driver_attr, const globusxio_contact-
t contactinfo, globusxio_operationnt op)
typedef globusresultt( globusxio_driver_serverdestroyt )(void driver_server)
typedef globusresultt( globusxio_driver_serveracceptt )(void driver_server, globusxio_operationnt op)
typedef globusresultt( globusxio_driver_servercntl_t )(void driver_server, int cmd, valist ap)
typedef globusresultt( globusxio_driver_link_destroyt )(void driver_link)
typedef globusresultt( globusxio_driver_transformopent )(const globusxio_contactt contactinfo, void
driver_link, void driver_attr, globusxio_operationnt op)
typedef globusresultt( globusxio_driver_transportopent )(const globusxio_contactt contactinfo, void
driver_link, void driver_attr, globusxio_operationnt op)
typedef globusresultt( globusxio_driver_handlecntl_t )(void handle, int cmd, valist ap)
typedef globusresultt( globusxio_driver_close_t )(void driver_handle, void driver_attr, globusxio-
operationnt op)
typedef globusresultt( globusxio_driver_readt )(void driver_speci c_handle, const globusxio_iovec-
t iovec, int ioveccount, globusxio_operationnt op)
typedef globusresultt( globusxio_driver_writet )(void driver_speci c_handle, const globusxio_iovec-
t iovec, int ioveccount, globusxio_operationnt op)
```

### Functions

```
globusresultt globusxio_driver_handlecntl (globusxio_driver_handlet driver_handle, globusxio_driver_-
t driver, int cmd,...)
void globusxio_driver_nished_accept(globusxio_operationnt op, void driver_link, globusresultt result)
globusresultt globusxio_driver_passopen(globusxio_operationnt op, const globusxio_contactt contact-
info, globusxio_driver_callbackt cb, void userarg)
void globusxio_driver_nished_open(void driver_handle, globusxio_operationnt op, globusresultt result)
globusresultt globusxio_driver_operationcreate (globusxio_operationnt operation, globusxio_driver_-
handle driver_handle)
globusboolt globusxio_driver_operationis_blocking(globusxio_operationnt op)
globusresultt globusxio_driver_passclose(globusxio_operationnt op, globusxio_driver_callbackt cb, void
userarg)
void globusxio_driver_nished_close(globusxio_operationnt op, globusresultt result)
globusresultt globusxio_driver_passread(globusxio_operationnt op, globusxio_iovec_t iovec, int iovec-
count, globussize_t wait_for, globusxio_driver_datacallbackt cb, void userarg)
void globusxio_driver_nished_read(globusxio_operationnt op, globusresultt result, globussize_t nread)
void globusxio_driver_seteof_received(globusxio_operationnt op)
```

```

globusbool_t globusxio_driver_eof_received(globusxio_operation_t op)
globusresult_t globusxio_driver_passwrite (globusxio_operation_t op, globusxio_iovec_t iovec, int iovec-
count, globussize_t wait_for, globusxio_driver_datacallback_t cb, void user_arg)
void globusxio_driver_nished_write (globusxio_operation_t op, globusresult_t result, globussize_t nwritten)
globusresult_t globusxio_driver_mergeoperation(globusxio_operation_t top_op, globusxio_operation_t bot-
tom_op)

```

#### 6.4.1 Detailed Description

The set of interface functions that the driver author must implement to create a driver and the functions to assist in the creation.

Driver attribute functions

If the driver wishes to provide driver specific attributes to the user it must implement the following functions:

```

globusxio_driver_attr_init_t  globusxio_driver_attr_copy_t  globusxio_driver_attr_cntl_t  globusxio_driver_attr-
destroy_t

```

#### 6.4.2 Typedef Documentation

6.4.2.1 typedef void( globusxio\_driver\_callback\_t)( globusxio\_operation\_t op, globusresult\_t result, void user\_arg)

callback interface

This is the function signature of callbacks for the `globusxio_driver_open/close()`.

Parameters:

- op The operation structure associated with the open or the close requested operation. The driver should call the appropriate nished operation to clean up this structure.
- result The result of the requested data operation
- user\_arg The user pointer that is threaded through to the callback.

6.4.2.2 typedef void( globusxio\_driver\_data\_callback\_t)( globusxio\_operation\_t op, globusresult\_t result, globussize\_t nbytes, void user\_arg)

Data Callback interface.

This is the function signature of read and write operation callbacks.

Parameters:

- op The operation structure associated with the read or write operation request. The driver should call the appropriate nished operation when it receives this operation.
- result The result of the requested data operation
- nbytes the number of bytes read or written
- user\_arg The user pointer that is threaded through to the callback.

6.4.2.3 typedef globusresult\_t( globusxio\_driver\_attr\_init\_t)( void out\_driver\_attr)

Create a driver specific attribute.

The driver should implement this function to create a driver specific attribute and return it via the `out_driver_attr` parameter.

6.4.2.4 `typedef globus_result_t( globus_xio_driver_attr_copy_t)( void dst, void src)`

Copy a driver attr.

When this function is called the driver will create a copy of the attr in parameter src and place it in the parameter dst.

6.4.2.5 `typedef globus_result_t( globus_xio_driver_attr_destroy_t)( void driver_attr)`

Destroy the driver attr.

Clean up all resources associate with the attr.

6.4.2.6 `typedef globus_result_t( globus_xio_driver_attr_cntl_t)( void attr, int cmd, va_list ap)`

get or set information in an attr.

The cmd parameter determines what functionality the user is requesting. The driver is responsible for providing documentation to the user on all the possible values that cmd can be.

Parameters:

driver\_attr The driver specific attr, created by `globus_xio_driver_attr_init_t`.

cmd An integer representing what functionality the user is requesting.

ap variable arguments. These are determined by the driver and the value of cmd.

6.4.2.7 `typedef globus_result_t( globus_xio_driver_server_init_t)( void driver_attr, const globus_xio_contact_t contact_info, globus_xio_operation_t op)`

Initialize a server object.

The driver developer should implement this function if their driver handles server operations (passive opens). In the tcp driver this function should create a listener.

Parameters:

op An op which should be passed to `globus_xio_driver_server_created`. Note, that unlike most operations, the server is created from the bottom of the stack to the top.

contact\_info This the contact info for the stack below this driver. (entries will always be NULL for the transport driver)

driver\_attr A server attr if the user specified any driver specific attributes. This may be NULL.

Returns:

Returning GLOBUSSUCCESS for this means that `globus_xio_driver_passserverinit` returned success and the driver's server was successfully initialized.

6.4.2.8 `typedef globus_result_t( globus_xio_driver_server_destroy_t)( void driver_server)`

destroy a server.

When this function is called the driver should free up all resources associated with a server.

Parameters:

server The server that the driver should clean up.

driver\_server The reference to the internal server that is being declared invalid with this function call.

6.4.2.9 `typedef globus_result_t( globus_xio_driver_server_accept_t)( void driver_server, globus_xio_operation_t op)`

Accept a server connection.

The driver developer should implement this function if their driver handles server operations. Once the accept operation completes, the connection is established. The user still has an opportunity to open the link or destroy it. They can query the link for additional information on which to base the decision to open.

Parameters:

`driver_server` The server object from which the link connection will be accepted.

`op` The requested operation. When the driver is finished accepting the server connection it uses this structure to signal globus\_xio that it has completed the operation.

6.4.2.10 `typedef globus_result_t( globus_xio_driver_server_ctrl_t)( void driver_server, int cmd, va_list ap)`

Query a server for information.

This function allows a user to request information from a driver specific server handle.

Parameters:

`driver_server` the server handle.

`cmd` An integer telling the driver what operation to perform on this server handle.

`ap` variable args.

6.4.2.11 `typedef globus_result_t( globus_xio_driver_link_destroy_t)( void driver_link)`

destroy a link

The driver should clean up all resources associated with the link when this function is called.

Parameters:

`driver_link` The link to be destroyed.

6.4.2.12 `typedef globus_result_t( globus_xio_driver_transform_open_t)( const globus_xio_contact_t contact_info, void driver_link, void driver_attr, globus_xio_operation_t op)`

Open a handle.

This is called when a user requests to open a handle.

Parameters:

`driver_link` Comes from server accept. Used to link an accepted connection to an xio handle. xio will destroy this object upon the return of this interface call.

`driver_attr` A attribute describing how to open. This points to a piece of memory created by the `globus_driver_driver_attr_init_t` interface function.

`contactinfo` Contains information about the requested resource. Its members may all be null (especially when link is not null). XIO will destroy this contact info upon return from the interface function

`op` The requested operation. When the driver is finished opening the handle it uses this structure to signal globus xio that it has completed the operation requested. It does this by calling `globus_xio_driver_finished_open()`

6.4.2.13 `typedef globus_result_t( globus_xio_driver_transport_open_t)( const globus_xio_contact_t contact_info, void driver_link, void driver_attr, globus_xio_operation_t op)`

transport open

6.4.2.14 `typedef globus_result_t( globus_xio_driver_handle_cntl_t)( void handle, int cmd, va_list ap)`

this call must return an `GLOBUSXIO_ERRORCOMMAND` error for unsupported command numbers.

(use `GlobusXIOErrorInvalidCommand(cmd)`)

Drivers that have reason to support the commands listed at [The globusxio user API](#) should accept the xio generic cmd numbers and their driver specific command number. Do NOT implement those handle cntls unless you really are the definitive source.

6.4.2.15 `typedef globus_result_t( globus_xio_driver_close_t)( void driver_handle, void driver_attr, globus_xio_operation_t op)`

Close an open handle.

This is called when a user requests to close a handle. The driver implementor should clean up all resources connected to there driver handle when this function is called.

Parameters:

`driver_specic_handle` The driver handle to be closed.

`driver_attr` A driver specific attr which may be used to alter how a close is performed (e.g, caching drivers)

`op` The requested operation. When the driver is finished closing the handle it uses this structure to signal globus xio that it has completed the operation requested. It does this by calling `globus_xio_driver_finished_close()`

6.4.2.16 `typedef globus_result_t( globus_xio_driver_read_t)( void driver_specic_handle, const globus_xio_iovec_t iovec, int iovec_count, globus_xio_operation_t op)`

Read data from an open handle.

This function is called when the user requests to read data from a handle. The driver author shall implement all code needed to for there driver to complete a read operations.

Parameters:

`driver_handle` The driver handle from which data should be read.

`iovec` An io vector pointing to the buffers to be read into.

`iovec_count` The number if entries in the io vector.

`op` The requested operation. When the driver is finished full lling the requested read operation it must use this structure to signal globus xio that the operation is completed. This is done by calling `globus_xio_driver_finished_operation()`..

6.4.2.17 `typedef globus_result_t( globus_xio_driver_write_t)( void driver_specic_handle, const globus_xio_iovec_t iovec, int iovec_count, globus_xio_operation_t op)`

Write data from an open handle.

This function is called when the user requests to write data to a handle. The driver author shall implement all code needed to for there driver to complete write operations.

## Parameters:

`driver_handle` The driver handle to which data should be written.

`iovec` An io vector pointing to the buffers to be written.

`iovec.count` The number of entries in the io vector.

`op` The requested operation. When the driver is finished fulfilling the requested read operation it must use this structure to signal globusxio that the operation is completed. This is done by calling `globusxio_driver_finished_operation()`.

## 6.4.3 Function Documentation

6.4.3.1 `globusresult_t globus_xio_driver_handle_cntl (globus_xio_driver_handle_t handle, globus_xio_driver_t driver, int cmd, ...)`

Touch driver specific information in a handle object.

pass the driver to control a specific driver pass NULL for driver for XIO specific cntls pass `GLOBUS_XIO_QUERY` for driver to try each driver (below current) in order

6.4.3.2 `void globusxio_driver_finished_accept (globusxio_operation_t op, void *driver_link, globusresult_t result)`

Driver API finished accept.

This function should be called to signal globusxio that it has completed the accept operation requested of it. It will free up resources associated with the accept and potentially cause xio to pop the signal up the driver stack.

## Parameters:

`op` The requested accept operation that has completed.

`driver_link` This is the initialized driver link that is that will be passed to the open interface when this handle is opened.

`result` Return status of the completed operation

6.4.3.3 `globusresult_t globus_xio_driver_passopen (globusxio_operation_t op, const globusxio_contact_t contactinfo, globus_xio_driver_callback_t cb, void *user_arg)`

Driver API Open.

This function will pass an open request down the driver stack. Upon completion of the open operation globusxio will call the `cb` function, at which point the handle structure will be initialized and available for use.

As soon as the function returns the handle is valid for creating other operations.

## Parameters:

`op` The operation from which the handle will be established. This parameter is used to determine what drivers are in the stack and other such information.

`contactinfo` The contact info describing the resource the driver below should open. This will normally be the same contact info that was passed in on the open interface.

`cb` The function to be called when the open operation is complete.

`user_arg` a user pointer that will be threaded through to the callback.

6.4.3.4 `void globusxio_driver_nished_open (void driver_handle, globusxio_operation_t op, globus_result_t result)`

Driver API nished open.

This function should be called to signal `globusxio` that it has completed the open operation requested of it. It will free up resources associated with the op and potentially cause xio to pop the signal up the driver stack.

Parameters:

- `driver_handle` The driver specific handle pointer that will be passed to future interface function calls.
- `op` The requested open operation that has completed.
- `result` Return status of the completed operation

6.4.3.5 `globus_result_t globusxio_driver_operation_create (globusxio_operation_t operation, globusxio_driver_handle_t handle)`

Driver API Create Operation.

This function will create an operation from an initialized handle. This operation can then be used for io operations related to the handle that created them.

Parameters:

- `operation` The operation to be created. When this function returns this structure will be populated and available for use for the driver.
- `handle` The initialized handle representing the user handle from which the operation will be created.

6.4.3.6 `globus_bool_t globusxio_driver_operation_is_blocking (globusxio_operation_t operation)`

Is Operation blocking.

If the operation is blocking the driver developer may be able to make certain optimizations. The function returns true if the given operation was created via a user call to a blocking function.

6.4.3.7 `globus_result_t globusxio_driver_passclose (globusxio_operation_t op, globusxio\_driver\_callback\_t cb, void user_arg)`

Driver API Close.

This function will pass a close request down the driver stack. Upon completion of the close operation `globusxio` will call the function pointed to by the cb argument.

Parameters:

- `op` The operation to pass along the driver stack for closing.
- `cb` A pointer to the function to be called once all drivers lower in the stack have closed.
- `user_arg` A user pointer that will be threaded through to the callback.

6.4.3.8 `void globusxio_driver_nished_close (globusxio_operation_t op, globus_result_t result)`

Driver API nished close.

The driver calls this function after completing a close operation on a `driver_handle`. Once this function returns the `driver_handle` is no longer valid.



## Parameters:

- op The close operation that has completed.
- result Return status of the completed operation

6.4.3.9 `globus_result_t globus_xio_driver_passread (globus_xio_operation_t op, globus_xio_iovec_t iovect, int iovect_count, globus_size_t wait_for, globus\_xio\_driver\_data\_callback\_t cb, void user_arg)`

## Driver read.

This function passes a read operation down the driver stack. After this function is called the op structure is no longer valid. However when the driver stack finishes servicing the read request it will pass a new operation structure in the function pointed to by cb. Finished read can be called on the new operation received.

## Parameters:

- op The operation structure representing this requested io operation.
- iovec A pointer to the array of iovecs.
- iovec\_count The number of iovecs in the array.
- wait\_for The minimum number of bytes to read before returning... if a driver has no specific requirement, he should use the user's request... available via `GlobusXIOOperationMinimumRead(op)`
- cb The function to be called when the operation request is completed.
- user\_arg A user pointer that will be threaded through to the callback.

6.4.3.10 `void globus_xio_driver_nished_read (globus_xio_operation_t op, globus_result_t result, globus_size_t nread)`

## Finished Read.

This function is called to signal `globus_xio` that the requested read operation has been completed.

## Parameters:

- op The operation structure representing the requested read operation.
- result Return status of the completed operation
- nread The number of bytes read

6.4.3.11 `void globus_xio_driver_set_eof_received (globus_xio_operation_t op)`

## EOF state manipulation.

This function is used by drivers that allow multiple outstanding reads at a time. It can only be called on behalf of a read operation (while in the read interface call or the ~~pass~~ callback).

Typical use for this would be to hold a driver specific lock and call this when an internal eof has been received. The read operation this is called on behalf of must be finished with an eof error or the results are undefined.

In general, you should not have an eof flag in your driver. Use this call `globus\_xio\_driver\_eof\_received\(\)` instead.

This is necessary to support xio's automatic eof resetting. If your driver absolutely can not be read after an eof has been set, then you will need your own eof flag.

This call will typically only be used just before a ~~nished~~`nished_read()` call.

## Parameters:

- op The operation structure representing the requested read operation.

6.4.3.12 `globus_bool_t globus_xio_driver_eof_received (globus_xio_operation_t op)`

EOF state checking.

This function is used by drivers that allow multiple outstanding reads at a time. It can only be called on behalf of a read operation (while in the read interface call or the `pass` callback).

Typical use for this would be to hold a driver specific lock (the same one used when calling `globus_xio_driver_set_eof_received()`) and call this to see if an eof has been received. If so, the operation should immediately be finished with an eof error (do not return an eof error).

This call will typically only be used in the read interface call.

Parameters:

`op` The operation structure representing the requested read operation.

Returns:

`GLOBUS_TRUE` if eof received, `GLOBUS_FALSE` otherwise.

6.4.3.13 `globus_result_t globus_xio_driver_pass_write (globus_xio_operation_t op, globus_xio_iovec_t iovec, int iovec_count, globus_size_t wait_for, globus_xio_driver_data_callback_t cb, void user_arg)`

Driver write.

This function passes a write operation down the driver stack. After this function is called the `op` structure is no longer valid. However when the driver stack finishes servicing the write request it will pass a new operation structure in the function pointed to by `cb`. Finished write can be called on the new operation received.

Parameters:

`op` The operation structure representing this requested io operation.

`iovec` A pointer to the array of iovecs.

`iovec_count` The number of iovecs in the array.

`wait_for` The minimum number of bytes to write before returning... if a driver has no specific requirement, he should use the user's request... available via `GlobusXIOOperationMinimumWrite(op)`

`cb` The function to be called when the operation request is completed.

`user_arg` A user pointer that will be threaded through to the callback.

6.4.3.14 `void globus_xio_driver_finished_write (globus_xio_operation_t op, globus_result_t result, globus_size_t nwritten)`

Finished Write.

This function is called to signal `globus_xio` that the requested write operation has been completed.

Parameters:

`op` The operation structure representing the requested write operation.

`result` Return status of the completed operation

`nwritten` The number of bytes written

6.4.3.15 `globus_result_t globus_xio_driver_merge_operation (globus_xio_operation_t top_op, globus_xio_operation_t bottom_op)`

Finishes an operation and merge two op structures.

(XXX not implemented yet)

This function will join two operations together and signal `globus_xio` that it has completed. This is an advanced function. Most drivers will not require its use. This function takes an operation that was created by this driver and passed on to drivers lower on the stack and an operation that came in on the interface function (that has seen the top half of the stack) and joins them together. The purpose of this function is to join data descriptors that were prestaged and cached with those that have later come in at the users request. See the read ahead doc for more information.

Parameters:

`top_op` The operation that has seen the top part of the driver stack.

`bottom_op` The operation that has seen the bottom part of the driver stack.

(result is always success in this case.. if there is an error, use the other `finish()` call)

## 6.5 Driver Programming: String options

A driver can choose to expose parameters as in a string form.

Functions

```
globus_result_t globus_xio_string_cntl_bouncer(globus_xio_driver_attr_cntl_t cntl_func, void attr, int cmd,...)
globus_result_t globus_xio_string_cntl_bool (void attr, const char key, const char val, int cmd, globus_xio_driver_attr_cntl_t cntl_func)
globus_result_t globus_xio_string_cntl_oat (void attr, const char key, const char val, int cmd, globus_xio_driver_attr_cntl_t cntl_func)
globus_result_t globus_xio_string_cntl_int (void attr, const char key, const char val, int cmd, globus_xio_driver_attr_cntl_t cntl_func)
globus_result_t globus_xio_string_cntl_string (void attr, const char key, const char val, int cmd, globus_xio_driver_attr_cntl_t cntl_func)
globus_result_t globus_xio_string_cntl_int_int (void attr, const char key, const char val, int cmd, globus_xio_driver_attr_cntl_t cntl_func)
```

### 6.5.1 Detailed Description

A driver can choose to expose parameters as in a string form.

Providing this feature makes dynamically setting driver specific options much easier. a user can then load the driver by name and set specific options by name all at runtime with no object module references. For example, a TCP driver can be loaded with the string: `tcp`, and the options can be set with:

```
port=50668keepalive=yes::nodelay=N
```

this would set the port to 50668, keepalive to true and nodelay to false. The particular string definition is defined by the tcp driver by properly creating a `globus_xio_attr_parsetable_t` array. Each element of the array is 1 options. There are 3 members of each array entry: key, cmd, and parse function. The key is a string that defines what option is to be set. In the above example string "port" would be 1 key. cmd tells the driver what cntl is associated with the key. In other words, once the string is parsed out what driver specific control must be called to set the requested option. For more information on controls see [Attributes and Cntls](#). The final value in the array entry is the parsing function. The parsing function takes the value of the `<key>=<value>` portion of the string and parses it into data types. once parsed

globus\_xio\_attr\_cntl is called and thus the option is set. There are many available parsing functions but the developer is free to right their own if the provided ones are not sufficient. Sample parsing functions follow:

[Driver Programming: String options](#)  
[Driver Programming: String options](#)  
[Driver Programming: String options](#)  
[Driver Programming: String options](#)  
[Driver Programming: String options](#)

### 6.5.2 Function Documentation

6.5.2.1 globusresult\_t globus\_xio\_string\_cntl\_bouncer ([globus\\_xio\\_driver\\_attr\\_cntl\\_t](#) cntl\_func, void attr, int cmd, ...)

New type functions call this one.

6.5.2.2 globusresult\_t globus\_xio\_string\_cntl\_bool (void attr, const char key, const char val, int cmd, [globus\\_xio\\_driver\\_attr\\_cntl\\_t](#) cntl\_func)

String option parsing function.

6.5.2.3 globusresult\_t globus\_xio\_string\_cntl\_oat (void attr, const char key, const char val, int cmd, [globus\\_xio\\_driver\\_attr\\_cntl\\_t](#) cntl\_func)

String option parsing function.

6.5.2.4 globusresult\_t globus\_xio\_string\_cntl\_int (void attr, const char key, const char val, int cmd, [globus\\_xio\\_driver\\_attr\\_cntl\\_t](#) cntl\_func)

String option parsing function.

6.5.2.5 globusresult\_t globus\_xio\_string\_cntl\_string (void attr, const char key, const char val, int cmd, [globus\\_xio\\_driver\\_attr\\_cntl\\_t](#) cntl\_func)

String option parsing function.

6.5.2.6 globusresult\_t globus\_xio\_string\_cntl\_int\_int (void attr, const char key, const char val, int cmd, [globus\\_xio\\_driver\\_attr\\_cntl\\_t](#) cntl\_func)

String option parsing function.

## 6.6 Globus XIO File Driver

The File I/O driver.

### Modules

[Opening/Closing](#)  
[Reading/Writing](#)  
[Env Variables](#)

Attributes and Cntls  
Types  
Error Types

### 6.6.1 Detailed Description

The File I/O driver.

## 6.7 Opening/Closing

An XIO handle with the file driver can be created with `globusxio_handlecreate()`

If there is no handle set on the attr passed to `globusxio_open()` call, it performs the equivalent of an `open()` call. In this case, the contact string must contain either a pathname or one of `stdin://`, `stdout://`, or `stderr://`. If a pathname is used, that path is opened. If one of the schemes are used the corresponding stdio handle is used (retrieved with `leno()`).

In either of the above cases, it is most efficient to call the blocking version `globusxio_open()`. It is also safe to call within a locked critical section.

When the XIO handle is closed, the file driver will destroy its internal resources and close the fd (unless this fd was set on an attr or converted from one of the stdio handles).

## 6.8 Reading/Writing

Both the `globusxio_registerread()` and `globusxio_registerwrite()` calls follow similar semantics as described below.

If the `waitforbytes` parameter is greater than zero, the io will happen asynchronously and be completed when at least `waitforbytes` has been read/written.

If the `waitforbytes` parameter is equal to zero, one of the following alternative behaviors occur:

If the length of the buffer is  $\leq 0$  the read or write happens synchronously. If the user is using one of the blocking xio calls, no internal callback will occur.

If the length of the buffer is also 0, the call behaves like an asynchronous notification of data ready to be either read or written. ie, an asynchronous `select()`.

In any case, when an error or EOF occurs before the `waitforbytes` request has been met, the outgoing `nbytes` is set to the amount of data actually read/written before the error or EOF occurred.

You may either use `GLOBUSXIO_FILE_SEEK` or `GLOBUSXIO_SEEK` to position the file pointer before each read or write or you can specify the desired offset on a data descriptor with the xio cmd, `GLOBUSXIO_CMD_SET_OFFSET`. simultaneous reading and writing is only predictable if the data descriptor method is used.

## 6.9 Env Variables

The file driver uses the following environment variables

`GLOBUSXIO_FILE_DEBUG` Available if using a debug build. See `globusxio.h` for format. The File driver defines the levels `TRACE` for all function call tracing and `INFO` for write buffer sizes

`GLOBUSXIO_SYSTEMDEBUG` Available if using a debug build. See `globusxio.h` for format. The File driver uses `globusxio.system` (along with the TCP and UDP drivers) which defines the following levels: `TRACE` for all function call tracing, `DATA` for data read and written counts, `INFO` for some special events,

and RAW which dumps the raw buffers actually read or written. This can contain binary data, so be careful when you enable it.

## 6.10 Attributes and Cntls

### Enumerations

```
enum globusxio_le_attr_cmd_t { GLOBUSXIO_FILE_SET_MODE, GLOBUSXIO_FILE_GET_MODE,
GLOBUSXIO_FILE_SET_FLAGS, GLOBUSXIO_FILE_GET_FLAGS, GLOBUSXIO_FILE_SET_
TRUNC_OFFSET, GLOBUSXIO_FILE_GET_TRUNC_OFFSET, GLOBUSXIO_FILE_SET_HANDLE,
GLOBUSXIO_FILE_GET_HANDLE, GLOBUSXIO_FILE_SET_BLOCKING_IO, GLOBUSXIO_FILE_
GET_BLOCKING_IO, GLOBUSXIO_FILE_SEEK}
```

### Functions

```
globusresult_t globusxio_attr_cntl (attr, driver, GLOBUSXIO_FILE_SET_MODE, int mode)
globusresult_t globusxio_attr_cntl (attr, driver, GLOBUSXIO_FILE_GET_MODE, int modeout)
globusresult_t globusxio_attr_cntl (attr, driver, GLOBUSXIO_FILE_SET_TRUNC_OFFSET, globusoff_t off-
set)
globusresult_t globusxio_attr_cntl (attr, driver, GLOBUSXIO_FILE_GET_TRUNC_OFFSET, globusoff_t off-
setout)
globusresult_t globusxio_attr_cntl (attr, driver, GLOBUSXIO_FILE_SET_HANDLE, globusxio_system_le_t
handle)
globusresult_t globusxio_attr_cntl (attr, driver, GLOBUSXIO_FILE_GET_HANDLE, globusxio_system-
le_t handleout)
globusresult_t globusxio_handle_cntl (handle, driver, GLOBUSXIO_FILE_GET_HANDLE, globusxio_-
system_le_t handleout)
globusresult_t globusxio_attr_cntl (attr, driver, GLOBUSXIO_FILE_SET_BLOCKING_IO, globusbool-
t useblockingio)
globusresult_t globusxio_handle_cntl (handle, driver, GLOBUSXIO_FILE_SET_BLOCKING_IO, globus-
bool_t useblockingio)
globusresult_t globusxio_attr_cntl (attr, driver, GLOBUSXIO_FILE_GET_BLOCKING_IO, globusbool-
t useblockingio_out)
globusresult_t globusxio_handle_cntl (handle, driver, GLOBUSXIO_FILE_GET_BLOCKING_IO, globus-
bool_t useblockingio_out)
globusresult_t globusxio_handle_cntl (handle, driver, GLOBUSXIO_FILE_SEEK, globusoff_t in_out_offset,
globusxio_le_whence_t whence)
```

#### 6.10.1 Detailed Description

File driver specific attrs and cntls.

See also:

[globusxio\\_attr\\_cntl\(\)](#), [globusxio\\_handle\\_cntl\(\)](#)

#### 6.10.2 Enumeration Type Documentation

##### 6.10.2.1 enum globusxio\_le\_attr\_cmd\_t

File driver specific cntls.

Enumeration values:

GLOBUS\_XIO\_FILE\_SET\_MODE See usage for [globusxio\\_attr\\_cntl](#).  
 GLOBUS\_XIO\_FILE\_GET\_MODE See usage for [globusxio\\_attr\\_cntl](#).  
 GLOBUS\_XIO\_FILE\_SET\_FLAGS See usage for [globusxio\\_attr\\_cntl](#).  
 GLOBUS\_XIO\_FILE\_GET\_FLAGS See usage for [globusxio\\_attr\\_cntl](#).  
 GLOBUS\_XIO\_FILE\_SET\_TRUNC\_OFFSET See usage for [globusxio\\_attr\\_cntl](#).  
 GLOBUS\_XIO\_FILE\_GET\_TRUNC\_OFFSET See usage for [globusxio\\_attr\\_cntl](#).  
 GLOBUS\_XIO\_FILE\_SET\_HANDLE See usage for [globusxio\\_attr\\_cntl](#).  
 GLOBUS\_XIO\_FILE\_GET\_HANDLE See usage for [globusxio\\_attr\\_cntl](#), [globusxio\\_handlecntl](#).  
 GLOBUS\_XIO\_FILE\_SET\_BLOCKING\_IO See usage for [globusxio\\_attr\\_cntl](#), [globusxio\\_handlecntl](#).  
 GLOBUS\_XIO\_FILE\_GET\_BLOCKING\_IO See usage for [globusxio\\_attr\\_cntl](#), [globusxio\\_handlecntl](#).  
 GLOBUS\_XIO\_FILE\_SEEK See usage for [globusxio\\_handlecntl](#).

### 6.10.3 Function Documentation

#### 6.10.3.1 `globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_SET_MODE, int mode)`

Set the file create mode.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Use this to set the permissions a non-existent file is created with, The default mode is 0644.

Parameters:

mode A bitwise OR of all the modes desired

See also:

[globusxio\\_file\\_modet](#)

string opt: mode<int>

#### 6.10.3.2 `globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_GET_MODE, int modeout)`

Get the file create mode.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

modeout The current mode will be stored here.

#### 6.10.3.3 `globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_SET_TRUNC_OFFSET, globus_off_t offset)`

Set the file truncate offset.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Use this in conjunction with the [types](#) tag to truncate a file to a non-zero offset. If the file was larger than offset bytes, the extra data is lost. If the file was shorter or non-existent, it is extended and the extended part reads as zeros. (default is 0)

Parameters:

offset The desired size of the le.

6.10.3.4 globusresult\_t globus\_xio\_attr\_cntl (attr, driver, GLOBUS\_XIO\_FILE\_GET\_TRUNC\_OFFSET, globus\_off\_t offset\_out)

Get the le truncate offset.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

offset\_out The offset will be stored here.

6.10.3.5 globusresult\_t globus\_xio\_attr\_cntl (attr, driver, GLOBUS\_XIO\_FILE\_SET\_HANDLE, globus\_xio\_system\_le\_t handle)

Set the le handle to use.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Do not open a new le, use this preopened handle instead.

Parameters:

handle Use this handle (fd or HANDLE) for the le. Note: close() will not be called on this handle.

6.10.3.6 globusresult\_t globus\_xio\_attr\_cntl (attr, driver, GLOBUS\_XIO\_FILE\_GET\_HANDLE, globus\_xio\_system\_le\_t handle\_out)

Get the le handle in use or in attr.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

handle\_out The le handle (fd or HANDLE) will be stored here. If none is set, GLOBUS\_XIO\_TCP\_INVALID - HANDLE will be set.

6.10.3.7 globusresult\_t globus\_xio\_handle\_cntl (handle, driver, GLOBUS\_XIO\_FILE\_GET\_HANDLE, globus\_xio\_system\_le\_t handle\_out)

Get the le handle in use or in attr.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

handle\_out The le handle (fd or HANDLE) will be stored here. If none is set, GLOBUS\_XIO\_TCP\_INVALID - HANDLE will be set.



6.10.3.8 `globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_SET_BLOCKING_IO, globus_bool_t use_blocking_io)`

Enable true blocking io when making `globus_xio_read/write()` calls.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Note: use with caution. you can deadlock an entire app with this.

Parameters:

`use_blocking_io` If `GLOBUS_TRUE`, true blocking io will be enabled. `GLOBUS_FALSE` will disable it (default);

6.10.3.9 `globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_FILE_SET_BLOCKING_IO, globus_bool_t use_blocking_io)`

Enable true blocking io when making `globus_xio_read/write()` calls.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Note: use with caution. you can deadlock an entire app with this.

Parameters:

`use_blocking_io` If `GLOBUS_TRUE`, true blocking io will be enabled. `GLOBUS_FALSE` will disable it (default);

6.10.3.10 `globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_GET_BLOCKING_IO, globus_bool_t use_blocking_io_out)`

Get the blocking io status in use or in attr.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`use_blocking_io_out` The flag will be set here. `GLOBUS_TRUE` for enabled.

string opt: `blocking=<bool>`

6.10.3.11 `globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_FILE_GET_BLOCKING_IO, globus_bool_t use_blocking_io_out)`

Get the blocking io status in use or in attr.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`use_blocking_io_out` The flag will be set here. `GLOBUS_TRUE` for enabled.

string opt: `blocking=<bool>`

6.10.3.12 `globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_FILE_SEEK, globus_off_t in_out_offset, globus_xio_le_whence_t whence)`

Reposition read/write le offset.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`in_out_offset` Specify the desired offset (according to whence). On success, the actual le offset will be stored here.

`whence` Specify how offset should be interpreted.

See also:

[globusxio\\_le\\_whencet](#), [GLOBUS\\_XIO\\_SEEK](#)

## 6.11 Types

De nes

#de ne [GLOBUS\\_XIO\\_FILE\\_INVALID\\_HANDLE](#)

Enumerations

```
enum globusxio_le_ag_t f GLOBUS\_XIO\_FILE\_CREAT = O_CREAT, GLOBUS\_XIO\_FILE\_EXCL = O_EXCL, GLOBUS\_XIO\_FILE\_RDONLY = O_RDONLY, GLOBUS\_XIO\_FILE\_WRONLY = O_WRONLY, GLOBUS\_XIO\_FILE\_RDWR = O_RDWR, GLOBUS\_XIO\_FILE\_TRUNC = O_TRUNC, GLOBUS\_XIO\_FILE\_APPEND = O_APPEND, GLOBUS\_XIO\_FILE\_BINARY = 0, GLOBUS\_XIO\_FILE\_TEXT = 0 g
enum globusxio_le_modet f GLOBUS\_XIO\_FILE\_IRWXU = S_IRWXU, GLOBUS\_XIO\_FILE\_IRUSR = S_IRUSR, GLOBUS\_XIO\_FILE\_IWUSR = S_IWUSR, GLOBUS\_XIO\_FILE\_IXUSR = S_IXUSR, GLOBUS\_XIO\_FILE\_IRWXO = S_IRWXO, GLOBUS\_XIO\_FILE\_IROTH = S_IROTH, GLOBUS\_XIO\_FILE\_IWOTH = S_IWOTH, GLOBUS\_XIO\_FILE\_IXOTH = S_IXOTH, GLOBUS\_XIO\_FILE\_IRWXG = S_IRWXG, GLOBUS\_XIO\_FILE\_IRGRP = S_IRGRP, GLOBUS\_XIO\_FILE\_IWGRP = S_IWGRP, GLOBUS\_XIO\_FILE\_IXGRP = S_IXGRP g
enum globusxio_le_whencet f GLOBUS\_XIO\_FILE\_SEEK\_SET = SEEK_SET, GLOBUS\_XIO\_FILE\_SEEK\_CUR = SEEK_CUR, GLOBUS\_XIO\_FILE\_SEEK\_END = SEEK_END g
```

### 6.11.1 De ne Documentation

#### 6.11.1.1 #de ne [GLOBUS\\_XIO\\_FILE\\_INVALID\\_HANDLE](#)

Invalid handle type.

See also:

[GLOBUS\\_XIO\\_FILE\\_SET\\_HANDLE](#)

### 6.11.2 Enumeration Type Documentation

#### 6.11.2.1 enum [globusxio\\_le\\_ag\\_t](#)

File driver open ags.

OR together all the ags you want

See also:

[GLOBUS\\_XIO\\_FILE\\_SET\\_FLAGS](#)

Enumeration values:

GLOBUS\_XIO\_FILE\_CREAT Create a new file if it doesn't exist (default).  
 GLOBUS\_XIO\_FILE\_EXCL Fail if file already exists.  
 GLOBUS\_XIO\_FILE\_RDONLY Open for read only.  
 GLOBUS\_XIO\_FILE\_WRONLY Open for write only.  
 GLOBUS\_XIO\_FILE\_RDWR Open for reading and writing (default).  
 GLOBUS\_XIO\_FILE\_TRUNC Truncate file.

See also:

[GLOBUS\\_XIO\\_FILE\\_SET\\_TRUNC\\_OFFSET](#)

GLOBUS\_XIO\_FILE\_APPEND Open file for appending.  
 GLOBUS\_XIO\_FILE\_BINARY File is binary (default).  
 GLOBUS\_XIO\_FILE\_TEXT File is text.

#### 6.11.2.2 enum globus\_xio\_file\_mode\_t

File driver create mode.

OR these modes together to get the mode you want.

See also:

[GLOBUS\\_XIO\\_FILE\\_SET\\_MODE](#)

NOTE: for Win32, you only have a choice between read-only and read-write. If the chosen mode does not specify writability, the file will be read only

Enumeration values:

GLOBUS\_XIO\_FILE\_IRWXU User read, write, and execute.  
 GLOBUS\_XIO\_FILE\_IRUSR User read.  
 GLOBUS\_XIO\_FILE\_IWUSR User write.  
 GLOBUS\_XIO\_FILE\_IXUSR User execute.  
 GLOBUS\_XIO\_FILE\_IRWXO Others read, write, and execute.  
 GLOBUS\_XIO\_FILE\_IROTH Others read.  
 GLOBUS\_XIO\_FILE\_IWOTH Others write.  
 GLOBUS\_XIO\_FILE\_IXOTH Others execute.  
 GLOBUS\_XIO\_FILE\_IRWXG Group read, write, and execute.  
 GLOBUS\_XIO\_FILE\_IRGRP Group read.  
 GLOBUS\_XIO\_FILE\_IWGRP Group write.  
 GLOBUS\_XIO\_FILE\_IXGRP Group execute.

#### 6.11.2.3 enum globus\_xio\_file\_whence\_t

File driver seek options.

See also:

[GLOBUS\\_XIO\\_FILE\\_SEEK](#)

Enumeration values:

GLOBUS\_XIO\_FILE\_SEEK\_SET set the file pointer at the specified offset  
 GLOBUS\_XIO\_FILE\_SEEK\_CUR set the file pointer at current position + offset  
 GLOBUS\_XIO\_FILE\_SEEK\_END set the file pointer at size of file + offset

## 6.12 Error Types

The File driver is very close to the system code, so most errors reported by it are converted from the system `errno`. A few of the exceptions are `GLOBUSXIO_ERROR_EOF`, `GLOBUSXIO_ERROR_COMMAND`, `GLOBUSXIO_ERROR_CONTACT_STRING`, and `GLOBUSXIO_ERROR_CANCELED`.

See also:

`globus_error_errno_match()`

## 6.13 Globus XIO HTTP Driver

This driver implements the HTTP/1.0 and HTTP/1.1 protocols within the Globus XIO framework.

### Modules

[Opening/Closing](#)  
[Reading/Writing](#)  
[Server](#)  
[Attributes and Cntls](#)  
[Error Types](#)

### Data Structures

`struct globusxio_http_header_t`  
 HTTP Header.

### Enumerations

`enum globusxio_http_version_t` f , `GLOBUSXIO_HTTP_VERSION_1_0`, `GLOBUSXIO_HTTP_VERSION_1_1` g

#### 6.13.1 Detailed Description

This driver implements the HTTP/1.0 and HTTP/1.1 protocols within the Globus XIO framework.

It may be used with the `tcp` driver for the standard HTTP protocol stack, or may be combined with the `gsi` driver for a HTTPS implementation.

This implementation supports user-defined HTTP headers, persistent connections, and chunked transfer encoding.

#### 6.13.2 Enumeration Type Documentation

##### 6.13.2.1 `enum globusxio_http_version_t`

Valid HTTP versions, used with the `GLOBUSXIO_HTTP_ATTR_SET_REQUEST_HTTP_VERSION` attribute and the `GLOBUSXIO_HTTP_HANDLE_SET_RESPONSE_HTTP_VERSION` handle control.

Enumeration values:

`GLOBUSXIO_HTTP_VERSION_1_0` HTTP/1.0.  
`GLOBUSXIO_HTTP_VERSION_1_1` HTTP/1.1.

## 6.14 Opening/Closing

An XIO handle with the http driver can be created with either `globusxio_handlecreate()` or `globusxio_serverregisteraccept()`.

If the handle is created with `globusxio_serverregisteraccept()` then an HTTP service handle will be created when `globusxio_registeropen()` is called. The XIO application must call one of the functions in the `globusxio_read()` family to receive the HTTP request metadata. This metadata will be returned in the data descriptor associated with that `rst` read: the application should use the `GLOBUSXIO_HTTP_GET_REQUEST` descriptor `cntl` to extract this metadata.

If the handle is created with `globusxio_handlecreate()` then an HTTP client handle will be created when `globusxio_registeropen()` is called. HTTP request headers, version and method may be chosen by setting attributes.

## 6.15 Reading/Writing

The HTTP driver behaves similar to the underlying transport driver with respect to reads and writes with the exception that metadata must be passed to the handle via open attributes on the client side and will be received as data descriptors as part of the `rst` request read or response read.

## 6.16 Server

The `globusxio_servercreate()` causes a new transport-specific listener socket to be created to handle new HTTP connections. `globusxio_serverregisteraccept()` will accept a new connection for processing. `globusxio_serverregisterclose()` cleans up the internal resources associated with the http server and calls `close` on the listener.

Multiple HTTP requests may be read in sequence from an HTTP server. After each request is processed and the response is sent (either by writing the entire entity body as specified by the Content-Length header or by using the `GLOBUSXIO_HTTP_HANDLE_SET_END_OF_ENTITY` handle `cntl`), the next read will contain the metadata related to the next operation. Only one request will be in process at once—the previous request must have sent or received and EOF (whichever is applicable to the request type).

## 6.17 Attributes and Cntls

### Enumerations

```
enum globusxio_http_handlecmd_t { GLOBUSXIO_HTTP_HANDLE_SET_RESPONSEHEADER,
GLOBUSXIO_HTTP_HANDLE_SET_RESPONSESTATUSCODE, GLOBUSXIO_HTTP_HANDLE_
SET_RESPONSEREASONPHRASE, GLOBUSXIO_HTTP_HANDLE_SET_RESPONSEHTTP_
VERSION, GLOBUSXIO_HTTP_HANDLE_SET_END_OF_ENTITY } g
enum globusxio_http_attr_cmd_t { GLOBUSXIO_HTTP_ATTR_SET_REQUESTMETHOD, GLOBUS_
XIO_HTTP_ATTR_SET_REQUESTHTTP_VERSION, GLOBUSXIO_HTTP_ATTR_SET_REQUEST_
HEADER, GLOBUSXIO_HTTP_ATTR_DELAY_WRITE_HEADER, GLOBUSXIO_HTTP_GET_
REQUEST, GLOBUSXIO_HTTP_GET_RESPONSE } g
```

### Functions

```
globusresult_t globusxio_handlecntl (handle, driver, GLOBUSXIO_HTTP_HANDLE_SET_RESPONSE-
HEADER, const char headename, const char headervalue)
globusresult_t globusxio_handlecntl (handle, driver, GLOBUSXIO_HTTP_HANDLE_SET_RESPONSE-
STATUSCODE, int status)
globusresult_t globusxio_handlecntl (handle, driver, GLOBUSXIO_HTTP_HANDLE_SET_RESPONSE-
REASONPHRASE, const char reason)
```

`globusresultt globusxio\_handlectl (handle, driver, GLOBUSXIO_HTTP_HANDLE_SET_RESPONSE_HTTP_VERSION, globusxio\_http\_versiont version)`  
`globusresultt globusxio\_handlectl (handle, driver, GLOBUSXIO_HTTP_HANDLE_SET_END_OF_ENTITY)`  
`globusresultt globusxio\_attr\_ctl (attr, driver, GLOBUSXIO_HTTP_ATTR_SET_REQUESTMETHOD, const char method)`  
`globusresultt globusxio\_attr\_ctl (attr, driver, GLOBUSXIO_HTTP_ATTR_SET_REQUESTHTTP_VERSION, globusxio\_http\_versiont version)`  
`globusresultt globusxio\_attr\_ctl (attr, driver, GLOBUSXIO_HTTP_ATTR_SET_REQUESTHEADER, const char headename, const charheadervalue)`

### 6.17.1 Detailed Description

HTTP driver specific attrs and cntls.

See also:

[globusxio\\_attr\\_ctl\(\)](#), [globusxio\\_handlectl\(\)](#)

### 6.17.2 Enumeration Type Documentation

#### 6.17.2.1 enum globusxio\_http\_handle\_cmd\_t

HTTP driver specific cntls.

Enumeration values:

`GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSEHEADER` See usage for [globusxio\\_handlectl](#).  
`GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSESTATUS_CODE` See usage for [globusxio\\_handlectl](#).  
`GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSEREASON.PHRASE` See usage for: [globusxio\\_handlectl](#).  
`GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSEHTTP_VERSION` See usage for [globusxio\\_handlectl](#).  
`GLOBUS_XIO_HTTP_HANDLE_SET_END_OF_ENTITY` See usage for [globusxio\\_handlectl](#).

#### 6.17.2.2 enum globusxio\_http\_attr\_cmd\_t

HTTP driver specific attribute and data descriptor cntls.

Enumeration values:

`GLOBUS_XIO_HTTP_ATTR_SET_REQUEST.METHOD` See usage for [globusxio\\_attr\\_ctl](#).  
`GLOBUS_XIO_HTTP_ATTR_SET_REQUEST.HTTP_VERSION` See usage for [globusxio\\_attr\\_ctl](#).  
`GLOBUS_XIO_HTTP_ATTR_SET_REQUEST.HEADER` See usage for [globusxio\\_attr\\_ctl](#).  
`GLOBUS_XIO_HTTP_ATTR_DELAY_WRITE_HEADER` See usage for [globusxio\\_attr\\_ctl](#).  
`GLOBUS_XIO_HTTP_GET_REQUEST` See usage for [globusxio\\_data\\_descriptorctl](#).  
`GLOBUS_XIO_HTTP_GET_RESPONSE` See usage for [globusxio\\_data\\_descriptorctl](#).

## 6.17.3 Function Documentation

6.17.3.1 `globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSEHEADER, const char headername, const char headervalue)`

Set the value of a response HTTP header.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

headername Name of the HTTP header to set.

headervalue Value of the HTTP header

Certain headers will cause changes in how the HTTP protocol will be handled. These include:

**Transfer-Encoding: identity|chunked** Override the default transfer encoding. If a server knows the exact length of the message body, or does not intend to support persistent connections, it may set this header to be "identity".

If this is set to "identity" and any of the following are true, then the connection will be closed after the end of the response is sent:

- A Content-Length header is not present
- The HTTP version is set to "HTTP/1.0"
- The Connection header is set to "close" Attempts to set this to "chunked" with an "HTTP/1.0" client will fail with a `GLOBUS_XIO_ERROR_HTTP_INVALID_HEADER` error.

**Content-Length: 1Digit**

- Provide a content length for the response message. If the "chunked" transfer encoding is being used, then this header will be silently ignored by the HTTP driver.

**Connection: close**

- The HTTP connection will be closed after the end of the data response is written.

Returns:

This handle control function can fail with

`GLOBUS_XIO_ERROR_MEMORY`  
`GLOBUS_XIO_ERROR_PARAMETER`  
`GLOBUS_XIO_ERROR_HTTP_INVALID_HEADER`

6.17.3.2 `globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSESTATUSCODE, int status)`

Set the response status code.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

status Value in the range 100-599 which will be used as the HTTP response code, as per RFC 2616.

If this cntl is not called by a server, then the default value of 200 ("Ok") will be used. If this is called on the client-side of an HTTP connection, the handle control will fail with a `GLOBUS_XIO_ERROR_PARAMETER` error.

Returns:

This handle control function can fail with

`GLOBUS_XIO_ERROR_PARAMETER`

6.17.3.3 `globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_REASON_PHRASE, const char reason)`

Set the response reason phrase.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`reason` The value of the HTTP response string, as per RFC 2616.

If this cntl is not called by a server, then a default value based on the handle's response status code will be generated. If this is called on the client-side of an HTTP connection, the handle control will fail with a `GLOBUS_XIO_ERROR_PARAMETER` error.

Returns:

This handle control function can fail with

`GLOBUS_XIO_ERROR_MEMORY`  
`GLOBUS_XIO_ERROR_PARAMETER`

6.17.3.4 `globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_HTTP_VERSION, globus\_xio\_http\_version\_t version)`

Set the response HTTP version.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`version` The HTTP version to be used in the server response line.

If this cntl is not called by a server, then the default of `GLOBUS_XIO_HTTP_VERSION_1_1` will be used, though no HTTP/1.1 features (chunking, persistent connections, etc) will be assumed if the client request was an HTTP/1.0 request. If this is called on the client-side of an HTTP connection, the handle control will fail with `GLOBUS_XIO_ERROR_PARAMETER`.

Returns:

This handle control function can fail with

`GLOBUS_XIO_ERROR_MEMORY`  
`GLOBUS_XIO_ERROR_PARAMETER`

6.17.3.5 `globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_HTTP_HANDLE_SET_END_OF_ENTITY)`

Indicate end-of-entity for an HTTP body.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

HTTP clients and servers must call this command to indicate to the driver that the entity-body which is being sent is completed. Subsequent attempts to write data on the handle will fail.

This handle command **MUST** be called on the client side of an HTTP connection when the HTTP method is `OPTIONS`, `POST`, or `PUT`, or when the open attributes indicate that an entity will be sent. This handle command **MUST** be called on the server side of an HTTP request connection when the HTTP method was `OPTIONS`, `GET`, `POST`, or `TRACE`.



6.17.3.6 `globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_HTTP_ATTR_SET_REQUEST_METHOD, const char method)`

Set the HTTP method to use for a client request.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`method` The request method string ("GET", "PUT", "POST", etc) that will be used in the HTTP request.

If this is not set on the target before it is opened, it will default to GET.

This attribute is ignored when opening the server side of an HTTP connection.

Setting this attribute may fail with

`GLOBUS_XIO_ERRORMEMORY`  
`GLOBUS_XIO_ERRORPARAMETER`

6.17.3.7 `globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_HTTP_ATTR_SET_REQUEST_HTTP_VERSION, globus\_xio\_http\_version\_t version)`

Set the HTTP version to use for a client request.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`version` The HTTP version to use for the client request.

If the client is using HTTP/1.0 in a request which will send a request message body (such as a POST or PUT), then the client MUST set the "Content-Length" HTTP header to be the length of the message. If this attribute is not present, then the default of `GLOBUS_XIO_HTTP_VERSION_1_1` will be used.

This attribute is ignored when opening the server side of an HTTP connection.

6.17.3.8 `globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_HTTP_ATTR_SET_REQUEST_HEADER, const char headername, const char headervalue)`

Set the value of an HTTP request header.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`headername` Name of the HTTP header to set.

`headervalue` Value of the HTTP header

Certain headers will cause the HTTP driver to behave differently than normal. This must be called before

`Transfer-Encoding: identity; chunked` Override the default transfer encoding. If a server knows the exact length of the message body, or does not intend to support persistent connections, it may set this header to be "identity".

If this is set to "identity" and any of the following are true, then the connection will be closed after the end of the message is sent:

- A Content-Length header is not present
- The HTTP version is set to "HTTP/1.0"
- The Connection header is set to "close" Attempts to set this to "chunked" with an "HTTP/1.0" client will fail with a GLOBUS\_XIO\_ERROR\_HTTP\_INVALID\_HEADER error.

Content-Length: 1Digit

- Provide a content length for the response message. If the "chunked" transfer encoding is being used, then this header will be silently ignored by the HTTP driver.

Connection: close

- If present in the server response, the connection will be closed after the end of the data response is written. Otherwise, when persistent connections are enabled, the connection is left open by the driver. Persistent connections are not yet implemented.

## 6.18 Error Types

### Enumerations

```
enum globusxio_http_errorst { GLOBUS_XIO_HTTP_ERROR_INVALID_HEADER, GLOBUS_XIO_HTTP_ERROR_PARSE, GLOBUS_XIO_HTTP_ERROR_NO_ENTITY, GLOBUS_XIO_HTTP_ERROR_EOF, GLOBUS_XIO_HTTP_ERROR_PERSISTENT_CONNECTION_DROPPED }
```

#### 6.18.1 Detailed Description

In addition to errors generated by underlying protocol drivers, the XIO HTTP driver defines a few error conditions specific to the HTTP protocol.

See also:

[globusxio\\_driver\\_error\\_match\(\)](#)

#### 6.18.2 Enumeration Type Documentation

##### 6.18.2.1 enum globusxio\_http\_errors\_t

Error types used to generate errors using the [globus](#) generic module.

Enumeration values:

GLOBUS\_XIO\_HTTP\_ERROR\_INVALID\_HEADER An attempt to set a header which is not compatible with the HTTP version being used.

GLOBUS\_XIO\_HTTP\_ERROR\_PARSE Error parsing HTTP protocol.

GLOBUS\_XIO\_HTTP\_ERROR\_NO\_ENTITY There is no entity body to read or write.

GLOBUS\_XIO\_HTTP\_ERROR\_EOF Server side fake EOF.

GLOBUS\_XIO\_HTTP\_ERROR\_PERSISTENT\_CONNECTION\_DROPPED Persistent connection dropped by the server.

## 6.19 Globus XIO MODE\_E Driver

### Modules

[Opening/Closing](#)

- Reading/Writing
- Server
- Env Variables
- Attributes and Cntls
- Types
- Error Types

## 6.20 Opening/Closing

An XIO handle with the mode `driver` can be created with either `globusxio_handlecreate()` or `globusxio_server_registeraccept()`

If the handle is created with `glibusxio_handlecreate()` the contact string passed to `glibusxio_registeropen()` call must contain a host name and service/port. The number of streams required can be specified on the attr using `Attributes and Control` (default is one stream). The stack of drivers to be used on the streams can be specified on the attr using `Attributes and Control` (default is a stack containing TCP driver).

When the XIO handle is closed, the model driver will destroy its internal resources and close the stream(s).

### 6.21 Reading/Writing

Mode E is unidirectional. Clients can only write and the server can only read. The `gpio_registerread()` enforce that the `waitforbytes` parameter should be one. When multiple transport streams are used between the client and the server, data might not be delivered in order. `gpio_data_descriptorctl()` can be used to get the offset of the data.

`globusxio_registerwrite()` does not enforce any restriction on the `waitforbytes` parameter.

In any case, when an error or EOF occurs before the waitforbytes request has been met, the outgoing nbytes is set to the amount of data actually read/written before the error or EOF occurred.

## 6.22 Server

`globusxio_servercreate()` causes a mode listener to be created and listened upon. `globusxio_serverregister-accept()` performs an asynchronous accept(). `globusxio_serverregisterclose()` cleans up the internal resources associated with the mode server.

All accepted handles inherit all mode specific attributes set in the attribute `ioctx` of the `ioctx` returned by `ioctx_create()`.

## 6.23 Env Variables

The modee driver uses the following environment variable

GLOBAL\_XIO\_MODE\_E\_DEBUG Available if using a debug build. See `globaldebug.h` for format.

## 6.24 Attributes and Cntls

## Enumerations

```
enum globusxio_mode_e cmd_t f  GLOBUS_XIO_MODE_E_SET_STACK,  GLOBUS_XIO_MODE_E_
GET_STACK,  GLOBUS_XIO_MODE_E_SET_NUM_STREAMS,  GLOBUS_XIO_MODE_E_GET_NUM_
STREAMS,  GLOBUS_XIO_MODE_E_SET_OFFSET_READS,  GLOBUS_XIO_MODE_E_GET_OFFSET_
```

READS, [GLOBUS\\_XIO\\_MODE\\_E\\_SET\\_MANUAL\\_EODC](#), [GLOBUS\\_XIO\\_MODE\\_E\\_GET\\_MANUAL\\_EODC](#), [GLOBUS\\_XIO\\_MODE\\_E\\_SEND\\_EOD](#), [GLOBUS\\_XIO\\_MODE\\_E\\_SET\\_EODC](#), [GLOBUS\\_XIO\\_MODE\\_E\\_DD\\_GET\\_OFFSET](#), [GLOBUS\\_XIO\\_MODE\\_E\\_SET\\_STACK\\_ATTR](#), [GLOBUS\\_XIO\\_MODE\\_E\\_GET\\_STACK\\_ATTR](#) g

## Functions

`globusresultt globusxio\_attr\_cntl (attr, driver, GLOBUSXIO_MODE_E_SET_STACK, globusxio_stack_t stack)`  
`globusresultt globusxio\_attr\_cntl (attr, driver, GLOBUSXIO_MODE_E_GET_STACK, globusxio_stack_t stackout)`  
`globusresultt globusxio\_attr\_cntl (attr, driver, GLOBUSXIO_MODE_E_SET_NUM_STREAMS, int num_streams)`  
`globusresultt globusxio\_attr\_cntl (attr, driver, GLOBUSXIO_MODE_E_GET_NUM_STREAMS, int num_streamsout)`  
`globusresultt globusxio\_attr\_cntl (attr, driver, GLOBUSXIO_MODE_E_SET_OFFSET_READS, globus_bool_t offset_reads)`  
`globusresultt globusxio\_attr\_cntl (attr, driver, GLOBUSXIO_MODE_E_GET_OFFSET_READS, globus_bool_t offset_readsout)`  
`globusresultt globusxio\_datadescriptor\_cntl (dd, driver, GLOBUSXIO_MODE_E_SEND_EOD, globus_bool_t sendeod)`  
`globusresultt globusxio\_handle\_cntl (handle, driver, GLOBUSXIO_MODE_E_SET_EODC, int eodcount)`  
`globusresultt globusxio\_datadescriptor\_cntl (dd, driver, GLOBUSXIO_MODE_E_DD_GET_OFFSET, globus_off_t offsetout)`  
`globusresultt globusxio\_attr\_cntl (attr, driver, GLOBUSXIO_MODE_E_GET_STACK_ATTR, globusxio_attr_t stackout)`

### 6.24.1 Detailed Description

Mode.e driver speci c attrs and cntls.

See also:

[globusxio\\_attr\\_cntl\(\)](#), [globusxio\\_handle\\_cntl\(\)](#), [globusxio\\_server\\_cntl\(\)](#), [globusxio\\_datadescriptor\\_cntl\(\)](#)

### 6.24.2 Enumeration Type Documentation

#### 6.24.2.1 enum globusxio\_mode.e.cmd\_t

MODE.E driver speci c cntls.

Enumeration values:

`GLOBUS_XIO_MODE_E_SET_STACK` See usage for [globusxio\\_attr\\_cntl](#).  
`GLOBUS_XIO_MODE_E_GET_STACK` See usage for [globusxio\\_attr\\_cntl](#).  
`GLOBUS_XIO_MODE_E_SET_NUM_STREAMS` See usage for [globusxio\\_attr\\_cntl](#).  
`GLOBUS_XIO_MODE_E_GET_NUM_STREAMS` See usage for [globusxio\\_attr\\_cntl](#).  
`GLOBUS_XIO_MODE_E_SET_OFFSET_READS` See usage for [globusxio\\_attr\\_cntl](#).  
`GLOBUS_XIO_MODE_E_GET_OFFSET_READS` See usage for [globusxio\\_attr\\_cntl](#).  
`GLOBUS_XIO_MODE_E_SET_MANUAL_EODC` See usage for [globusxio\\_attr\\_cntl](#).  
`GLOBUS_XIO_MODE_E_GET_MANUAL_EODC` See usage for [globusxio\\_attr\\_cntl](#).

GLOBUS\_XIO\_MODE\_E\_SEND\_EOD See usage for [globusxio\\_data\\_descriptorctl](#).  
 GLOBUS\_XIO\_MODE\_E\_SET\_EODC See usage for [globusxio\\_handlectl](#).  
 GLOBUS\_XIO\_MODE\_E\_DD\_GET\_OFFSET See usage for [globusxio\\_data\\_descriptorctl](#).  
 GLOBUS\_XIO\_MODE\_E\_SET\_STACK\_ATTR See usage for [globusxio\\_attrctl](#).  
 GLOBUS\_XIO\_MODE\_E\_GET\_STACK\_ATTR See usage for [globusxio\\_attrctl](#).

### 6.24.3 Function Documentation

6.24.3.1 `globus_result_t globus_xio_attr_ctl (attr, driver, GLOBUS_XIO_MODE_E_SET_STACK, globus_xio_stack_t stack)`

Set the stack (of xio drivers) to be used for the connection(s).

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Do not create a new ftp client handle, use this handle instead.

Parameters:

`stack` Specifies the stack to use for the connection(s). Note: this stack will not be destroyed.

6.24.3.2 `globus_result_t globus_xio_attr_ctl (attr, driver, GLOBUS_XIO_MODE_E_GET_STACK, globus_xio_stack_t stackout)`

Get the stack on the attr.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`stackout` The stack will be stored here. If none is set, `GLOBUS_XIO_STACK_NULL` will be set.

6.24.3.3 `globus_result_t globus_xio_attr_ctl (attr, driver, GLOBUS_XIO_MODE_E_SET_NUM_STREAMS, int num_streams)`

Set the number of streams to be used between the client and the server.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`num_streams` Specifies the number of streams to use.

6.24.3.4 `globus_result_t globus_xio_attr_ctl (attr, driver, GLOBUS_XIO_MODE_E_GET_NUM_STREAMS, int num_streamsout)`

Get the number of streams on the attr.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`num_streamsout` The stream count will be stored here.

6.24.3.5 `globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_MODE_E_SET_OFFSET_READS, globus_bool_t offsetreads)`

Set `ag` to indicate whether the data read from user would always be preceded by an offset read or not.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

The user can do a read with `wait_bytes` set to zero, to find the offset of the data that he is going to get in his next read operation

Parameters:

`offsetreads` `GLOBUS_TRUE` to enable offset reads, `GLOBUS_FALSE` to disable offset reads (default).

6.24.3.6 `globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_MODE_E_GET_OFFSET_READS, globus_bool_t offsetreadsout)`

Get `OFFSET_READS` `ag` on the `attr`.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`offsetreadsout` The `OFFSET_READS` `ag` will be stored here.

6.24.3.7 `globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_MODE_E_SEND_EOD, globus_bool_t sendeod)`

Set `SENDEOD` `ag`.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used only for data descriptors to write calls.

Parameters:

`sendeod` `GLOBUS_TRUE` to send EOD, `GLOBUS_FALSE` to not send EOD (default).

6.24.3.8 `globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_MODE_E_SET_EODC, int eodcount)`

Set EOD count.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used only if `MANUAL_EODC` `ag` is set to `GLOBUS_TRUE`.

Parameters:

`eodcount` specifies the eod count

6.24.3.9 `globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_MODE_E_DD_GET_OFFSET, globus_off_t offsetout)`

Get offset of the next available data.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used only if OFFSETREADS is enabled.

Parameters:

offsetout offset will be stored here

6.24.3.10 `globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_MODE_E_GET_STACK_ATTR, globus_xio_attr_t stackout)`

Get the attr that will be used with the stack.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is intended for use with `GLOBUS_XIO_MODE_E_SET_STACK`.

Parameters:

stackout The stack will be stored here. If none is set, `GLOBUS_XIO_MODE_E_SET_STACK` will be set.

## 6.25 Types

### 6.26 Error Types

Enumerations

`enum globus_xio_mode_e_error_type_t` `GLOBUS_XIO_MODE_E_HEADER_ERROR`

#### 6.26.1 Detailed Description

The errors reported by `MODE` driver include `GLOBUS_XIO_ERROR_COMMAND`, `GLOBUS_XIO_ERROR_MEMORY`, `GLOBUS_XIO_ERROR_STATE`, `GLOBUS_XIO_ERROR_PARAMETER`, `GLOBUS_XIO_ERROR_EOF`, `GLOBUS_XIO_ERROR_CANCELED`, [Error Types](#)

See also:

[globus\\_xio\\_driver\\_error\\_match\(\)](#), [globus\\_error\\_errno\\_match\(\)](#)

#### 6.26.2 Enumeration Type Documentation

##### 6.26.2.1 `enum globus_xio_mode_e_error_type_t`

`MODE_E` driver specific error types.

Enumeration values:

`GLOBUS_XIO_MODE_E_HEADER_ERROR` Indicates that the mode header is erroneous.

## 6.27 Globus XIO ORDERING Driver

Modules

[Opening/Closing](#)

[Reading/Writing](#)  
[Env Variables](#)  
[Attributes and Cntls](#)  
[Types](#)  
[Error Types](#)

## 6.28 Opening/Closing

Ordering driver is a transform driver and thus has to be used on top of a transport driver. An XIO handle with the ordering driver can be created with either [globusxio\\_handlecreate\(\)](#) or [globusxio\\_serverregisteraccept\(\)](#)

When the XIO handle is closed, the ordering driver will destroy its internal resources.

## 6.29 Reading/Writing

Ordering driver does not allow multiple [globusxio\\_registerread\(\)](#) to be outstanding. This limitation is there to enforce that the users get the read callback in order. There is a known issue in enforcing the order in which read callbacks are delivered with multiple outstanding reads. This limitation does not restrict the use of parallel reads feature provided by the underlying transport driver. [Attributes and Cntls](#) on the attr can be used to specify the number of parallel reads. Ordering will have a maximum of this many number of reads outstanding to the driver below it on the stack. It buffers the data read and delivers it to the user in order.

[globusxio\\_registerwrite\(\)](#) does not enforce any restriction.

## 6.30 Env Variables

The ordering driver uses the following environment variable

GLOBUS.XIO\_ORDERINGDEBUG Available if using a debug build. See [globusdebug.h](#) for format.

## 6.31 Attributes and Cntls

Enumerations

```
enum globusxio_orderingcmd_t { GLOBUS_XIO_ORDERING_SET_OFFSET, GLOBUS_XIO_ORDERING_SET_MAX_READ_COUNT, GLOBUS_XIO_ORDERING_GET_MAX_READ_COUNT, GLOBUS_XIO_ORDERING_SET_BUFFERING, GLOBUS_XIO_ORDERING_GET_BUFFERING, GLOBUS_XIO_ORDERING_SET_BUF_SIZE, GLOBUS_XIO_ORDERING_GET_BUF_SIZE, GLOBUS_XIO_ORDERING_SET_MAX_BUF_COUNT, GLOBUS_XIO_ORDERING_GET_MAX_BUF_COUNT };
```

Functions

```
globus_result_t globusxio_handlecntl (handle, driver, GLOBUS_XIO_ORDERING_SET_OFFSET, globus_off_t offset)
globus_result_t globusxio_attr.cntl (attr, driver, GLOBUS_XIO_ORDERING_SET_MAX_READ_COUNT, int max_readcount)
globus_result_t globusxio_attr.cntl (attr, driver, GLOBUS_XIO_ORDERING_GET_MAX_READ_COUNT, int max_readcountout)
globus_result_t globusxio_attr.cntl (attr, driver, GLOBUS_XIO_ORDERING_SET_BUFFERING, globus_bool_t buffering)
```



globus\_result\_t [globus\\_xio\\_attr\\_cntl](#) (attr, driver, GLOBUS\_XIO\_ORDERING\_GET\_BUFFERING, globus\_bool\_t buffering\_out)

### 6.31.1 Detailed Description

Ordering driver specific attrs and cntls.

See also:

[globus\\_xio\\_attr\\_cntl\(\)](#), [globus\\_xio\\_handlecntl\(\)](#)

### 6.31.2 Enumeration Type Documentation

#### 6.31.2.1 enum globus\_xio\_ordering\_cmd\_t

ORDERING driver specific cntls.

Enumeration values:

GLOBUS\_XIO\_ORDERING\_SET\_OFFSET See usage for [globus\\_xio\\_handlecntl](#).  
 GLOBUS\_XIO\_ORDERING\_SET\_MAX\_READ\_COUNT See usage for [globus\\_xio\\_attr\\_cntl](#).  
 GLOBUS\_XIO\_ORDERING\_GET\_MAX\_READ\_COUNT See usage for [globus\\_xio\\_attr\\_cntl](#).  
 GLOBUS\_XIO\_ORDERING\_SET\_BUFFERING See usage for [globus\\_xio\\_attr\\_cntl](#).  
 GLOBUS\_XIO\_ORDERING\_GET\_BUFFERING See usage for [globus\\_xio\\_attr\\_cntl](#).  
 GLOBUS\_XIO\_ORDERING\_SET\_BUF\_SIZE See usage for [globus\\_xio\\_attr\\_cntl](#).  
 GLOBUS\_XIO\_ORDERING\_GET\_BUF\_SIZE See usage for [globus\\_xio\\_attr\\_cntl](#).  
 GLOBUS\_XIO\_ORDERING\_SET\_MAX\_BUF\_COUNT See usage for [globus\\_xio\\_attr\\_cntl](#).  
 GLOBUS\_XIO\_ORDERING\_GET\_MAX\_BUF\_COUNT See usage for [globus\\_xio\\_attr\\_cntl](#).

### 6.31.3 Function Documentation

#### 6.31.3.1 globus\_result\_t globus\_xio\_handle\_cntl (handle, driver, GLOBUS\_XIO\_ORDERING\_SET\_OFFSET, globus\_off\_t offset)

Set offset for the next IO operation.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is not allowed when there is an outstanding IO operation. This operation clears all the buffered data.

Parameters:

offset Specifies the offset to use in the next IO operation.

#### 6.31.3.2 globus\_result\_t globus\_xio\_attr\_cntl (attr, driver, GLOBUS\_XIO\_ORDERING\_SET\_MAX\_READ\_COUNT, int max\_read\_count)

Set the maximum number of reads that ordering driver can have outstanding on driver(s) below.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

max\_read\_count Specifies the maximum number of parallel reads (default is 1).

6.31.3.3 `globusresult_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_ORDERING_GET_MAX_READ_COUNT, int max_read_count_out)`

Get the maximum number of parallel reads set on the attr.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`max_read_count_out` The maximum number of parallel reads allowed will be stored here.

6.31.3.4 `globusresult_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_ORDERING_SET_BUFFERING, globus_bool_t buffering)`

This driver can be used in 2 modes; ordering (care about offsets of the data read - underlying transport driver may deliver data out of order - this driver will rearrange data based on the offset and deliver in order to user) and buffering (do not care about offsets - just buffer the data read and deliver it when requested).

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This attribute control can be used to enable buffering.

Parameters:

`buffering` `GLOBUS_TRUE` to enable buffering, `GLOBUS_FALSE` (default) to disable buffering.

6.31.3.5 `globusresult_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_ORDERING_GET_BUFFERING, globus_bool_t buffering_out)`

Get the buffering flag on the attr.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`buffering_out` Buffering flag will be stored in here.

## 6.32 Types

### 6.33 Error Types

Enumerations

```
enum globusxio_orderingerror_type_t { GLOBUS_XIO_ORDERING_ERROR_READ, GLOBUS_XIO_ORDERING_ERROR_CANCEL }
```

#### 6.33.1 Detailed Description

The errors reported by ORDERING driver include `GLOBUS_XIO_ERROR_COMMAND`, `GLOBUS_XIO_ERROR_MEMORY`, `GLOBUS_XIO_ERROR_STATE`, `GLOBUS_XIO_ERROR_CANCELED`

See also:

[globusxio\\_driver\\_error\\_match\(\)](#), [globuserror\\_errno\\_match\(\)](#)

### 6.33.2 Enumeration Type Documentation

#### 6.33.2.1 enum globusxio\_ordering\_error\_type\_t

ORDERING driver specific error types.

Enumeration values:

    GLOBUS\_XIO\_ORDERING\_ERROR\_READ Indicates that an error occurred in reading data.

    GLOBUS\_XIO\_ORDERING\_ERROR\_CANCEL Indicates an error occurred in canceling an operation.

## 6.34 Globus XIO TCP Driver

The IPV4/6 TCP socket driver.

Modules

- [Opening/Closing](#)
- [Reading/Writing](#)
- [Server](#)
- [Env Variables](#)
- [Attributes and Cntls](#)
- [Types](#)
- [Error Types](#)

### 6.34.1 Detailed Description

The IPV4/6 TCP socket driver.

## 6.35 Opening/Closing

An XIO handle with the tcp driver can be created with either [globusxio\\_handlecreate\(\)](#) or [globusxio\\_serverregister-accept\(\)](#).

If the handle is created with [globusxio\\_serverregisteraccept\(\)](#) the [globusxio\\_registeropen\(\)](#) call does nothing more than initialize the internal handle with the accepted socket.

If the handle is created with [globusxio\\_handlecreate\(\)](#) and there is no handle set on the attr passed to [globusxio\\_registeropen\(\)](#) call, it performs the equivalent of an asynchronous connect() call. In this case, the contact string must contain a host name and service/port. Both the hostname and port number can be numeric or symbolic (eg: some.webserver.com:80 or 214.123.12.1:http). If the hostname is symbolic and it resolves to multiple ip addresses, each one will be attempted in succession, until the connect is successful or there are no more addresses.

When the XIO handle is closed, the tcp driver will destroy its internal resources and close the socket (unless this socket was set on an attr). Any write data pending in system buffers will be sent unless the linger option has been set. Any remaining data in recv buffers will be discarded and (on some systems) a connection reset sent to the peer.

## 6.36 Reading/Writing

Both the [globusxio\\_registerread\(\)](#) and [globusxio\\_registerwrite\(\)](#) calls follow similar semantics as described below.

If the waitforbytes parameter is greater than zero, the io will happen asynchronously and be completed when at least waitforbytes has been read/written.

If the `waitforbytes` parameter is equal to zero, one of the following alternative behaviors occur:

If the length of the buffer is  $\geq 0$  the read or write happens synchronously. If the user is using one of the blocking xio calls, no internal callback will occur.

If the length of the buffer is also 0, the call behaves like an asynchronous notification of data ready to be either read or written. ie, an asynchronous `select()`.

In any case, when an error or EOF occurs before the `waitforbytes` request has been met, the outgoing `nbytes` is set to the amount of data actually read/written before the error or EOF occurred.

## 6.37 Server

`globusxio_servercreate()` causes a tcp listener socket to be created and listened upon. `globusxio_serverregister-accept()` performs an asynchronous accept. `globusxio_serverregisterclose()` cleans up the internal resources associated with the tcp server and calls `close()` on the listener socket (unless the socket was set on the server via the `attr`)

All accepted handles inherit all tcp specific attributes set in the `attr` to `globusxio_servercreate()` but can be overridden with the `attr` to `globusxio_registeropen()`

## 6.38 Env Variables

The tcp driver uses the following environment variables

**GLOBUS\_HOSTNAME** Used when setting the hostname in the contact string

**GLOBUS\_TCP\_PORT\_RANGE** Used to restrict anonymous listener ports ex: `GLOBUS_TCP_PORT_RANGE=4000,4100`

**GLOBUS\_TCP\_PORT\_RANGE\_STATE\_FILE** Used in conjunction with **GLOBUS\_TCP\_PORT\_RANGE** to maintain last used port among many applications making use of the same port range. That last port + 1 will be used as a starting point within the specified tcp port range instead of always starting at the beginning. This is really only necessary when a machine is behind a stateful firewall which is holding a port in a different state than the application's machine. See [bugzilla.globus.org](http://bugzilla.globus.org), bug 1851 for more info. ex: `GLOBUS_TCP_PORT_RANGE_STATE_FILE=/tmp/portstate` (file will be created if it does not exist)

**GLOBUS\_TCP\_SOURCE\_RANGE** Used to restrict local ports used in a connection

**GLOBUS\_XIO\_TCP\_DEBUG** Available if using a debug build. See `globusbug.h` for format. The TCP driver defines the levels TRACE for all function call tracing and INFO for write buffer sizes

**GLOBUS\_XIO\_SYSTEM\_DEBUG** Available if using a debug build. See `globusbug.h` for format. The TCP driver uses `globusxio_system` (along with the File and UDP drivers) which defines the following levels: TRACE for all function call tracing, DATA for data read and written counts, INFO for some special events, and RAW which dumps the raw buffers actually read or written. This can contain binary data, so be careful when you enable it.

## 6.39 Attributes and Cntls

Enumerations

```
enum globusxio_tcp_cmd_t {
    GLOBUS_XIO_TCP_SET_SERVICE, GLOBUS_XIO_TCP_GET_SERVICE,
    GLOBUS_XIO_TCP_SET_PORT, GLOBUS_XIO_TCP_GET_PORT, GLOBUS_XIO_TCP_SET_BACKLOG,
    GLOBUS_XIO_TCP_GET_BACKLOG, GLOBUS_XIO_TCP_SET_LISTEN_RANGE, GLOBUS_XIO_TCP_GET_LISTEN_RANGE,
    GLOBUS_XIO_TCP_GET_HANDLE, GLOBUS_XIO_TCP_SET_HANDLE,

```

GLOBUS\_XIO\_TCP\_SET\_INTERFACE, GLOBUS\_XIO\_TCP\_GET\_INTERFACE, GLOBUS\_XIO\_TCP\_SET\_RESTRICTPORT, GLOBUS\_XIO\_TCP\_GET\_RESTRICTPORT, GLOBUS\_XIO\_TCP\_SET\_REUSEADDR, GLOBUS\_XIO\_TCP\_GET\_REUSEADDR, GLOBUS\_XIO\_TCP\_SET\_NO\_IPV6, GLOBUS\_XIO\_TCP\_GET\_NO\_IPV6, GLOBUS\_XIO\_TCP\_SET\_CONNECT\_RANGE, GLOBUS\_XIO\_TCP\_GET\_CONNECT\_RANGE, GLOBUS\_XIO\_TCP\_SET\_KEEPALIVE, GLOBUS\_XIO\_TCP\_GET\_KEEPALIVE, GLOBUS\_XIO\_TCP\_SET\_LINGER, GLOBUS\_XIO\_TCP\_GET\_LINGER, GLOBUS\_XIO\_TCP\_SET\_OOBLIN, GLOBUS\_XIO\_TCP\_GET\_OOBLIN, GLOBUS\_XIO\_TCP\_SET\_SNDBUF, GLOBUS\_XIO\_TCP\_GET\_SNDBUF, GLOBUS\_XIO\_TCP\_SET\_RCVBUF, GLOBUS\_XIO\_TCP\_GET\_RCVBUF, GLOBUS\_XIO\_TCP\_SET\_NODELAY, GLOBUS\_XIO\_TCP\_GET\_NODELAY, GLOBUS\_XIO\_TCP\_SET\_SEND\_FLAGS, GLOBUS\_XIO\_TCP\_GET\_SEND\_FLAGS, GLOBUS\_XIO\_TCP\_GET\_LOCAL\_CONTACT, GLOBUS\_XIO\_TCP\_GET\_LOCAL\_NUMERIC\_CONTACT, GLOBUS\_XIO\_TCP\_GET\_REMOTE\_CONTACT, GLOBUS\_XIO\_TCP\_GET\_REMOTE\_NUMERIC\_CONTACT, GLOBUS\_XIO\_TCP\_AFFECT\_ATTR\_DEFAULTS, GLOBUS\_XIO\_TCP\_SET\_BLOCKING\_IO, GLOBUS\_XIO\_TCP\_GET\_BLOCKING\_IO

g

## Functions

globusresultt [globusxio\\_attr\\_cntl](#) (attr, driver, GLOBUSXIO\_TCP\_SET\_SERVICE, const char service-name)  
 globusresultt [globusxio\\_attr\\_cntl](#) (attr, driver, GLOBUSXIO\_TCP\_GET\_SERVICE, char servicename-out)  
 globusresultt [globusxio\\_attr\\_cntl](#) (attr, driver, GLOBUSXIO\_TCP\_SET\_PORT, int listeneport)  
 globusresultt [globusxio\\_attr\\_cntl](#) (attr, driver, GLOBUSXIO\_TCP\_GET\_PORT, int listenerport.out)  
 globusresultt [globusxio\\_attr\\_cntl](#) (attr, driver, GLOBUSXIO\_TCP\_SET\_LISTEN\_RANGE, int listenemin-port, int listenemax.port)  
 globusresultt [globusxio\\_attr\\_cntl](#) (attr, driver, GLOBUSXIO\_TCP\_GET\_LISTEN\_RANGE, int listener-min.port.out, int listenermax.port.out)  
 globusresultt [globusxio\\_attr\\_cntl](#) (attr, driver, GLOBUSXIO\_TCP\_GET\_HANDLE, globusxio\_systemsocket handleout)  
 globusresultt [globusxio\\_handlecntl](#) (handle, driver, GLOBUSXIO\_TCP\_GET\_HANDLE, globusxio\_systemsocket handleout)  
 globusresultt [globusxio\\_servercntl](#) (server, driver, GLOBUSXIO\_TCP\_GET\_HANDLE, globusxio\_systemsocket handleout)  
 globusresultt [globusxio\\_attr\\_cntl](#) (attr, driver, GLOBUSXIO\_TCP\_SET\_HANDLE, globusxio\_systemsocket handle)  
 globusresultt [globusxio\\_attr\\_cntl](#) (attr, driver, GLOBUSXIO\_TCP\_SET\_RESTRICTPORT, globusbool restrict.port)  
 globusresultt [globusxio\\_attr\\_cntl](#) (attr, driver, GLOBUSXIO\_TCP\_GET\_RESTRICTPORT, globusbool restrict.port.out)  
 globusresultt [globusxio\\_handlecntl](#) (handle, driver, GLOBUSXIO\_TCP\_SET\_KEEPALIVE, globusbool keepalive)  
 globusresultt [globusxio\\_handlecntl](#) (handle, driver, GLOBUSXIO\_TCP\_GET\_KEEPALIVE, globusbool keepalive.out)  
 globusresultt [globusxio\\_attr\\_cntl](#) (attr, driver, GLOBUSXIO\_TCP\_SET\_LINGER, globusbool linger, int linger.time)  
 globusresultt [globusxio\\_handlecntl](#) (handle, driver, GLOBUSXIO\_TCP\_SET\_LINGER, globusbool t linger, int lingertime)  
 globusresultt [globusxio\\_attr\\_cntl](#) (attr, driver, GLOBUSXIO\_TCP\_GET\_LINGER, globusbool linger.out, int linger.time.out)  
 globusresultt [globusxio\\_handlecntl](#) (handle, driver, GLOBUSXIO\_TCP\_GET\_LINGER, globusbool t linger.out, int linger.time.out)  
 globusresultt [globusxio\\_handlecntl](#) (handle, driver, GLOBUSXIO\_TCP\_SET\_SNDBUF, int sndbuf)

```

globusresultt globusxio\_handlecntl (handle, driver, GLOBUSXIO_TCP_GET_SNDBUF, int sndbufout)
globusresultt globusxio\_datadescriptorcntl (dd, driver, GLOBUSXIO_TCP_SET_SEND_FLAGS, int
send ags)
globusresultt globusxio\_datadescriptorcntl (dd, driver, GLOBUSXIO_TCP_GET_SEND_FLAGS, int
send ags_out)
globusresultt globusxio\_handlecntl (handle, driver, GLOBUSXIO_TCP_GET_LOCAL_CONTACT, char
contactstring.out)
globusresultt globusxio\_servercntl (server, driver, GLOBUSXIO_TCP_GET_LOCAL_CONTACT, char
contactstring.out)

```

### 6.39.1 Detailed Description

Tcp driver speci c attr and cntls.

See also:

[globusxio\\_attr.cntl\(\)](#), [globusxio\\_handlecntl\(\)](#), [globusxio\\_servercntl\(\)](#), [globusxio\\_datadescriptorcntl\(\)](#)

### 6.39.2 Enumeration Type Documentation

#### 6.39.2.1 enum globusxio\_tcp\_cmd\_t

TCP driver speci c cntls.

Enumeration values:

```

GLOBUS_XIO_TCP_SET_SERVICE See usage for globusxio\_attr.cntl.
GLOBUS_XIO_TCP_GET_SERVICE See usage for globusxio\_attr.cntl.
GLOBUS_XIO_TCP_SET_PORT See usage for globusxio\_attr.cntl.
GLOBUS_XIO_TCP_GET_PORT See usage for globusxio\_attr.cntl.
GLOBUS_XIO_TCP_SET_BACKLOG See usage for globusxio\_attr.cntl.
GLOBUS_XIO_TCP_GET_BACKLOG See usage for globusxio\_attr.cntl.
GLOBUS_XIO_TCP_SET_LISTEN_RANGE See usage for globusxio\_attr.cntl.
GLOBUS_XIO_TCP_GET_LISTEN_RANGE See usage for globusxio\_attr.cntl.
GLOBUS_XIO_TCP_GET_HANDLE See usage for globusxio\_attr.cntl, globusxio\_handlecntl, globus-
xio\_servercntl.
GLOBUS_XIO_TCP_SET_HANDLE See usage for globusxio\_attr.cntl.
GLOBUS_XIO_TCP_SET_INTERFACE See usage for globusxio\_attr.cntl.
GLOBUS_XIO_TCP_GET_INTERFACE See usage for globusxio\_attr.cntl.
GLOBUS_XIO_TCP_SET_RESTRICT_PORT See usage for globusxio\_attr.cntl.
GLOBUS_XIO_TCP_GET_RESTRICT_PORT See usage for globusxio\_attr.cntl.
GLOBUS_XIO_TCP_SET_REUSEADDR See usage for globusxio\_attr.cntl.
GLOBUS_XIO_TCP_GET_REUSEADDR See usage for globusxio\_attr.cntl.
GLOBUS_XIO_TCP_SET_NO_IPV6 See usage for globusxio\_attr.cntl.
GLOBUS_XIO_TCP_GET_NO_IPV6 See usage for globusxio\_attr.cntl.
GLOBUS_XIO_TCP_SET_CONNECT_RANGE See usage for globusxio\_attr.cntl.
GLOBUS_XIO_TCP_GET_CONNECT_RANGE See usage for globusxio\_attr.cntl.
GLOBUS_XIO_TCP_SET_KEEPALIVE See usage for globusxio\_attr.cntl, globusxio\_handlecntl.

```

GLOBUS\_XIO\_TCP\_GET\_KEEPALIVE See usage for [globusxio\\_attr\\_cntl](#), [globusxio\\_handlecntl](#).  
 GLOBUS\_XIO\_TCP\_SET\_LINGER See usage for [globusxio\\_attr\\_cntl](#), [globusxio\\_handlecntl](#).  
 GLOBUS\_XIO\_TCP\_GET\_LINGER See usage for [globusxio\\_attr\\_cntl](#), [globusxio\\_handlecntl](#).  
 GLOBUS\_XIO\_TCP\_SET\_OOBLINE See usage for [globusxio\\_attr\\_cntl](#), [globusxio\\_handlecntl](#).  
 GLOBUS\_XIO\_TCP\_GET\_OOBLINE See usage for [globusxio\\_attr\\_cntl](#), [globusxio\\_handlecntl](#).  
 GLOBUS\_XIO\_TCP\_SET\_SNDBUF See usage for [globusxio\\_attr\\_cntl](#), [globusxio\\_handlecntl](#).  
 GLOBUS\_XIO\_TCP\_GET\_SNDBUF See usage for [globusxio\\_attr\\_cntl](#), [globusxio\\_handlecntl](#).  
 GLOBUS\_XIO\_TCP\_SET\_RCVBUF See usage for [globusxio\\_attr\\_cntl](#), [globusxio\\_handlecntl](#).  
 GLOBUS\_XIO\_TCP\_GET\_RCVBUF See usage for [globusxio\\_attr\\_cntl](#), [globusxio\\_handlecntl](#).  
 GLOBUS\_XIO\_TCP\_SET\_NODELAY See usage for [globusxio\\_attr\\_cntl](#), [globusxio\\_handlecntl](#).  
 GLOBUS\_XIO\_TCP\_GET\_NODELAY See usage for [globusxio\\_attr\\_cntl](#), [globusxio\\_handlecntl](#).  
 GLOBUS\_XIO\_TCP\_SET\_SEND\_FLAGS See usage for [globusxio\\_data\\_descriptorcntl](#).  
 GLOBUS\_XIO\_TCP\_GET\_SEND\_FLAGS See usage for [globusxio\\_data\\_descriptorcntl](#).  
 GLOBUS\_XIO\_TCP\_GET\_LOCAL\_CONTACT See usage for [globusxio\\_handlecntl](#), [globusxio\\_servercntl](#).  
 GLOBUS\_XIO\_TCP\_GET\_LOCAL\_NUMERIC\_CONTACT See usage for: [globusxio\\_handlecntl](#), [globusxio\\_servercntl](#).  
 GLOBUS\_XIO\_TCP\_GET\_REMOTE\_CONTACT See usage for [globusxio\\_handlecntl](#).  
 GLOBUS\_XIO\_TCP\_GET\_REMOTE\_NUMERIC\_CONTACT See usage for [globusxio\\_handlecntl](#).  
 GLOBUS\_XIO\_TCP\_AFFECT\_ATTR\_DEFAULTS See usage for [globusxio\\_attr\\_cntl](#).  
 GLOBUS\_XIO\_TCP\_SET\_BLOCKING\_IO See usage for [globusxio\\_attr\\_cntl](#), [globusxio\\_handlecntl](#).  
 GLOBUS\_XIO\_TCP\_GET\_BLOCKING\_IO See usage for [globusxio\\_attr\\_cntl](#), [globusxio\\_handlecntl](#).

### 6.39.3 Function Documentation

6.39.3.1 `globusresult_t globusxio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_SERVICE, const char servicename)`

Set the tcp service name to bind to.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used only on attrs for [globusxio\\_servercreate\(\)](#)

Parameters:

`servicename` The service name to use when setting up the listener. If the service name cannot be resolved, the port (if one is set) will be used instead.

string opt: `port< string>`

6.39.3.2 `globusresult_t globusxio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_SERVICE, char service.name.out)`

Get the tcp service name to bind to.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`service.name.out` A pointer to the service name will be stored here. If none is set, NULL will be passed back. Otherwise, the name will be duplicated with `strdup()` and the user should call `free()` on it.



### 6.39.3.3 `globusresult_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_PORT, int listener.port)`

Set the tcp port number to bind to.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used only on attrs for [globus\\_xio\\_servercreate\(\)](#) The default port number is 0 (system assigned)

Parameters:

`listener.port` The port number to use when setting up the listener. If the service name is also set, this will only be used if that can't be resolved.

string opt: `port=<int>`

### 6.39.3.4 `globusresult_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_PORT, int listener.port.out)`

Get the tcp port number to bind to.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`listener.port.out` The port will be stored here.

### 6.39.3.5 `globusresult_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_LISTEN_RANGE, int listener.min_port, int listener.max_port)`

Set the tcp port range to connect the server to.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used only on attrs for [globus\\_xio\\_servercreate\(\)](#) where no specific service or port has been set. It overrides the range set in the `GLOBUS_TCP_PORT_RANGE` env variable. If 'restrict port' is true, the server's listening port will be constrained to the range specified.

Parameters:

`listener.min_port` The lower bound on the listener port. (default 0 – no bound)

`listener.max_port` The upper bound on the listener port. (default 0 – no bound)

See also:

[GLOBUS\\_XIO\\_TCP\\_SET\\_RESTRICTPORT](#)

string opt: `listenrange=<int>,<int>`

### 6.39.3.6 `globusresult_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_LISTEN_RANGE, int listener.min_port.out, int listener.max_port.out)`

Get the tcp port range on an attr.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.



## Parameters:

listener\_min\_port\_out The lower bound will be stored here.

listener\_max\_port\_out The upper bound will be stored here.

6.39.3.7 `globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_HANDLE, globus_xio_system_socket_t handle_out)`

Get the tcp socket handle on an attr, handle, or server.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

## Parameters:

handle\_out The tcp socket will be stored here. If none is set, `GLOBUS_XIO_TCP_INVALID_HANDLE` will be set.

6.39.3.8 `globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_GET_HANDLE, globus_xio_system_socket_t handle_out)`

Get the tcp socket handle on an attr, handle, or server.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

## Parameters:

handle\_out The tcp socket will be stored here. If none is set, `GLOBUS_XIO_TCP_INVALID_HANDLE` will be set.

6.39.3.9 `globus_result_t globus_xio_server_cntl (server, driver, GLOBUS_XIO_TCP_GET_HANDLE, globus_xio_system_socket_t handle_out)`

Get the tcp socket handle on an attr, handle, or server.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

## Parameters:

handle\_out The tcp socket will be stored here. If none is set, `GLOBUS_XIO_TCP_INVALID_HANDLE` will be set.

6.39.3.10 `globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_HANDLE, globus_xio_system_socket_t handle)`

Set the tcp socket to use for a handle or server.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used only on attrs for [globus\\_xio\\_servercreate\(\)](#) or [globus\\_xio\\_registeropen\(\)](#)

## Parameters:

handle Use this handle (fd or SOCKET) for the listener or connection. Note: `close()` will not be called on this handle.

6.39.3.11 `globusresult_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_RESTRICT_PORT, globus_bool_t restrict_port)`

Enable or disable the listener or connector range constraints.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used only on attrs for [globus\\_xio\\_servercreate\(\)](#) or [globus\\_xio\\_registeropen\(\)](#) This enables or ignores the port range found in the attr or in then env. By default, those ranges are enabled.

Parameters:

`restrict_port` GLOBUS\_TRUE to enable (default), GLOBUS\_FALSE to disable.

See also:

[GLOBUS\\_XIO\\_TCP\\_SET\\_LISTEN\\_RANGE](#), [GLOBUS\\_XIO\\_TCP\\_SET\\_CONNECT\\_RANGE](#)

6.39.3.12 `globusresult_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_RESTRICT_PORT, globus_bool_t restrict_port_out)`

Get the restrict port ag.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`restrict_port_out` The restrict port ag will be stored here.

6.39.3.13 `globusresult_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_SET_KEEPALIVE, globus_bool_t keepalive)`

Enable tcp keepalive.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used on attrs for [globus\\_xio\\_servercreate\(\)](#) [globus\\_xio\\_registeropen\(\)](#) and with [globus\\_xio\\_handlecntl\(\)](#) to determine whether or not to periodically send "keepalive" messages on a connected socket handle. This may enable earlier detection of broken connections.

Parameters:

`keepalive` GLOBUS\_TRUE to enable, GLOBUS\_FALSE to disable (default)

string opt: `keepalive=<bool>`

6.39.3.14 `globusresult_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_GET_KEEPALIVE, globus_bool_t keepaliveout)`

Get the tcp keepalive ag.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`keepaliveout` The tcp keepalive ag will be stored here.

6.39.3.15 `globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_LINGER, globus_bool_t linger, int linger_time)`

Set tcp linger.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used on attrs for `globus_xio_servercreate()` `globus_xio_registeropen()` and with `globus_xio_handlecntl()` to determine what to do when data is in the socket's buffer when the socket is closed. If `linger` is set to true, then the close operation will block until the socket buffers are empty, or the `linger_time` has expired. If this is enabled, any data remaining after the linger time has expired, will be discarded. If this is disabled, close nishes immediately, but the OS will still attempt to transmit the remaining data.

Parameters:

`linger` GLOBUS.TRUE to enable, GLOBUS.FALSE to disable (default)

`linger_time` The time (in seconds) to block at close time if `linger` is true and data is queued in the socket buffer.

6.39.3.16 `globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_SET_LINGER, globus_bool_t linger, int linger_time)`

Set tcp linger.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used on attrs for `globus_xio_servercreate()` `globus_xio_registeropen()` and with `globus_xio_handlecntl()` to determine what to do when data is in the socket's buffer when the socket is closed. If `linger` is set to true, then the close operation will block until the socket buffers are empty, or the `linger_time` has expired. If this is enabled, any data remaining after the linger time has expired, will be discarded. If this is disabled, close nishes immediately, but the OS will still attempt to transmit the remaining data.

Parameters:

`linger` GLOBUS.TRUE to enable, GLOBUS.FALSE to disable (default)

`linger_time` The time (in seconds) to block at close time if `linger` is true and data is queued in the socket buffer.

6.39.3.17 `globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_LINGER, globus_bool_t linger_out, int linger_time_out)`

Get the tcp linger ag and time.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`linger_out` The linger ag will be stored here.

`linger_time_out` The linger time will be set here.

6.39.3.18 `globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_GET_LINGER, globus_bool_t linger_out, int linger_time_out)`

Get the tcp linger ag and time.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

## Parameters:

linger\_out The linger ag will be stored here.

linger\_time\_out The linger time will be set here.

6.39.3.19 `globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_SET_SNDBUF, int sndbuf)`

Set the tcp socket send buffer size.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used on attrs for [globusxio\\_servercreate\(\)](#) [globusxio\\_registeropen\(\)](#) and with [globusxio\\_handlecntl\(\)](#) to set the size of the send buffer used on the socket.

## Parameters:

sndbuf The send buffer size in bytes to use. (default is system spec c)

string opt: sndbuf formatted in

6.39.3.20 `globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_GET_SNDBUF, int sndbuf_out)`

Get the tcp send buffer size on the attr or handle.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

## Parameters:

sndbuf\_out The send buffer size will be stored here.

6.39.3.21 `globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_TCP_SET_SEND_FLAGS, int send_ags)`

Set tcp send ags.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used only for data descriptors to write calls.

## Parameters:

send\_ags The ags to use when sending data.

See also:

[globusxio\\_tcp\\_send\\_ags\\_t](#)

6.39.3.22 `globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_TCP_GET_SEND_FLAGS, int send_ags_out)`

Get tcp send ags.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

## Parameters:

send\_ags\_out The ags to use will be stored here.

6.39.3.23 `globus_result_t globus_xio_handle_cntl` (`handle`, `driver`, `GLOBUS_XIO_TCP_GET_LOCAL_CONTACT`, `char contactstring_out`)

Get local socket info.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`contactstring_out` A pointer to a contact string for the local end of a connected socket or listener will be stored here. The user should `free()` it when done with it. It will be in the format `hostname:<port>`

See also:

[globus\\_xio\\_server\\_get\\_contactstring\(\)](#), [GLOBUS\\_XIO\\_GET\\_LOCAL\\_CONTACT](#)

6.39.3.24 `globus_result_t globus_xio_server_cntl` (`server`, `driver`, `GLOBUS_XIO_TCP_GET_LOCAL_CONTACT`, `char contactstring_out`)

Get local socket info.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`contactstring_out` A pointer to a contact string for the local end of a connected socket or listener will be stored here. The user should `free()` it when done with it. It will be in the format `hostname:<port>`

See also:

[globus\\_xio\\_server\\_get\\_contactstring\(\)](#), [GLOBUS\\_XIO\\_GET\\_LOCAL\\_CONTACT](#)

## 6.40 Types

Defines

`#define GLOBUS_XIO_TCP_INVALID_HANDLE`

Enumerations

`enum globus_xio_tcp_send_flags_t { GLOBUS_XIO_TCP_SEND_OOB = MSG_OOB }`

### 6.40.1 Define Documentation

#### 6.40.1.1 `#define GLOBUS_XIO_TCP_INVALID_HANDLE`

Invalid handle type.

See also:

[GLOBUS\\_XIO\\_TCP\\_SET\\_HANDLE](#)

### 6.40.2 Enumeration Type Documentation

#### 6.40.2.1 enum globusxio\_tcp\_send\_agst

TCP driver specific types.

Enumeration values:

`GLOBUS_XIO_TCP_SEND_OOB` Use this with [Attributes and Cntls](#) to send a TCP message out of band (Urgent data flag set).

## 6.41 Error Types

Enumerations

enum [globusxio\\_tcp\\_error\\_type\\_t](#) f `GLOBUS_XIO_TCP_ERROR_NO_ADDRS`g

### 6.41.1 Detailed Description

The TCP driver is very close to the system code, so most errors reported by it are converted from the system `errno`. A few of the exceptions are `GLOBUS_XIO_ERROR_EOF`, `GLOBUS_XIO_ERROR_COMMAND`, `GLOBUS_XIO_ERROR_CONTACT_STRING`, `GLOBUS_XIO_ERROR_CANCELED`, and [Error Types](#)

See also:

[globusxio\\_driver\\_error\\_match\(\)](#), [globuserror\\_errno\\_match\(\)](#)

### 6.41.2 Enumeration Type Documentation

#### 6.41.2.1 enum globusxio\_tcp\_error\_type\_t

TCP driver specific error types.

Enumeration values:

`GLOBUS_XIO_TCP_ERROR_NO_ADDRS` Indicates that no IPv4/6 compatible sockets could be resolved for the specified hostname.

## 6.42 Globus XIO UDP Driver

The IPV4/6 UDP socket driver.

Modules

[Opening/Closing](#)  
[Reading/Writing](#)  
[Env Variables](#)  
[Attributes and Cntls](#)  
[Types](#)  
[Error Types](#)

### 6.42.1 Detailed Description

The IPV4/6 UDP socket driver.

## 6.43 Opening/Closing

An XIO handle with the udp driver can be created with `globusxio_handlecreate()`

The handle can be created in two modes: open server or connected client. If the contact string does not have a host and port, the udp socket will accept messages from any sender. If a host and port is specified, the udp socket will be 'connected' immediately to that host:port. This blocks packets from any sender other than the contact string. A handle that starts out as an open server can later be 'connected' with `globusxio_handleconnect()` (presumably after the first message is received from a sender and his contact info is available).

When the XIO handle is closed, the udp driver will destroy its internal resources and close the socket (unless this socket was set on the attribute `globusxio_registeropen()`).

## 6.44 Reading/Writing

`globusxio_registerread()` semantics:

If the waitforbytes parameter is greater than zero, the read will happen asynchronously and be completed when at least waitforbytes has been read/written.

If the waitforbytes parameter is equal to zero, one of the following alternative behaviors occur:

If the length of the buffer is > 0 the read happens synchronously. If the user is using one of the blocking xio calls, no internal callback will occur.

If the length of the buffer is also 0, the call behaves like an asynchronous notification of data ready to be read. ie, an asynchronous select().

In any case, when an error occurs before the waitforbytes request has been met, the outgoing nbytes is set to the amount of data actually read before the error occurred.

If the handle is not connected, the user should pass in a data descriptor. After the read, this descriptor will contain the contact string of the sender. The user can either get this contact string with `globusxio_handlegetcontact()` or pass the data descriptor directly to `globusxio_registerwrite()` to send a message back to the sender.

Also, if the handle is not connected, the waitforbytes should probably be 1 to guarantee that only one packet is received and the sender contact isn't overwritten by multiple packets from different senders.

`globusxio_registerwrite()` semantics:

When performing a write, exactly one UDP packet is sent of the entire buffer length. The waitforbytes parameter is ignored. If the entire buffer can not be written, `Error Types` error will be returned with nbytes set to the number of bytes actually sent.

If the handle is not 'connected', a contact string must be set in the data descriptor with `globusxio_registerwrite()`. This can either be done explicitly with `globusxio_handlesetcontact()` or implicitly by passing in a data descriptor received from `globusxio_registerread()`

The udp write semantics are always synchronous. No blocking or internal callback will occur when `globusxio_write()`.

## 6.45 Env Variables

The udp driver uses the following environment variables

GLOBUS.HOSTNAME Used when setting the hostname in the contact string  
 GLOBUS.UDP.PORT.RANGE Used to restrict the port the udp socket binds to  
 GLOBUS.XIO.SYSTEM.DEBUG Available if using a debug build. See `globusxio_debug.h` for format. The UDP driver uses `globusxio_system` (along with the File and TCP drivers) which defines the following levels: TRACE

for all function call tracing, DATA for data read and written counts, INFO for some special events, and RAW which dumps the raw buffers actually read or written. This can contain binary data, so be careful when you enable it.

## 6.46 Attributes and Cntls

### Enumerations

```
enum globusxio_udp_cmd_t { GLOBUSXIO_UDP_SET_HANDLE, GLOBUSXIO_UDP_SET_SERVICE,
GLOBUSXIO_UDP_GET_SERVICE, GLOBUSXIO_UDP_SET_PORT, GLOBUSXIO_UDP_GET_PORT,
GLOBUSXIO_UDP_SET_LISTEN_RANGE, GLOBUSXIO_UDP_GET_LISTEN_RANGE, GLOBUSXIO_UDP_SET_INTERFACE,
GLOBUSXIO_UDP_GET_INTERFACE, GLOBUSXIO_UDP_SET_RESTRICT_PORT, GLOBUSXIO_UDP_GET_RESTRICT_PORT,
GLOBUSXIO_UDP_SET_REUSEADDR, GLOBUSXIO_UDP_GET_REUSEADDR, GLOBUSXIO_UDP_SET_NO_IPV6,
GLOBUSXIO_UDP_GET_NO_IPV6, GLOBUSXIO_UDP_GET_HANDLE, GLOBUSXIO_UDP_SET_SNDBUF,
GLOBUSXIO_UDP_GET_SNDBUF, GLOBUSXIO_UDP_SET_RCVBUF, GLOBUSXIO_UDP_GET_RCVBUF,
GLOBUSXIO_UDP_GET_CONTACT, GLOBUSXIO_UDP_GET_NUMERIC_CONTACT, GLOBUSXIO_UDP_SET_CONTACT,
GLOBUSXIO_UDP_CONNECT, GLOBUSXIO_UDP_SET_MULTICAST };
```

### Functions

```
globus_result_t globusxio_attr_cntl (attr, driver, GLOBUSXIO_UDP_SET_HANDLE, globusxio_system_socket_t handle)
globus_result_t globusxio_attr_cntl (attr, driver, GLOBUSXIO_UDP_SET_SERVICE, const char service_name)
globus_result_t globusxio_attr_cntl (attr, driver, GLOBUSXIO_UDP_GET_SERVICE, char service_name_out)
globus_result_t globusxio_attr_cntl (attr, driver, GLOBUSXIO_UDP_SET_PORT, int listener_port)
globus_result_t globusxio_attr_cntl (attr, driver, GLOBUSXIO_UDP_GET_PORT, int listener_port_out)
globus_result_t globusxio_attr_cntl (attr, driver, GLOBUSXIO_UDP_SET_LISTEN_RANGE, int listener_min_port, int listener_max_port)
globus_result_t globusxio_attr_cntl (attr, driver, GLOBUSXIO_UDP_GET_LISTEN_RANGE, int listener_min_port_out, int listener_max_port_out)
globus_result_t globusxio_attr_cntl (attr, driver, GLOBUSXIO_UDP_SET_RESTRICT_PORT, globus_bool_t restrict_port)
globus_result_t globusxio_attr_cntl (attr, driver, GLOBUSXIO_UDP_GET_RESTRICT_PORT, globus_bool_t restrict_port_out)
globus_result_t globusxio_attr_cntl (attr, driver, GLOBUSXIO_UDP_GET_HANDLE, globusxio_system_socket_t handle_out)
globus_result_t globusxio_handle_cntl (handle, driver, GLOBUSXIO_UDP_GET_HANDLE, globusxio_system_socket_t handle_out)
globus_result_t globusxio_handle_cntl (handle, driver, GLOBUSXIO_UDP_SET_SNDBUF, int sndbuf)
globus_result_t globusxio_handle_cntl (handle, driver, GLOBUSXIO_UDP_GET_SNDBUF, int sndbuf_out)
globus_result_t globusxio_handle_cntl (handle, driver, GLOBUSXIO_UDP_GET_CONTACT, char contact_string_out)
globus_result_t globusxio_data_descriptor_cntl (dd, driver, GLOBUSXIO_UDP_GET_CONTACT, char contact_string_out)
globus_result_t globusxio_data_descriptor_cntl (dd, driver, GLOBUSXIO_UDP_SET_CONTACT, char contact_string)
globus_result_t globusxio_handle_cntl (handle, driver, GLOBUSXIO_UDP_CONNECT, char contact_string)
globus_result_t globusxio_attr_cntl (attr, driver, GLOBUSXIO_UDP_SET_MULTICAST, char contact_string)
```



## 6.46.1 Detailed Description

UDP driver specific attrs and cntls.

See also:

[globusxio\\_attr.cntl\(\)](#), [globusxio\\_handlecntl\(\)](#), [globusxio\\_datadescriptorcntl\(\)](#)

## 6.46.2 Enumeration Type Documentation

## 6.46.2.1 enum globusxio\_udp\_cmd\_t

UDP driver specific cntls.

Enumeration values:

GLOBUS\_XIO\_UDP\_SET\_HANDLE See usage for [globusxio\\_attr.cntl](#).  
 GLOBUS\_XIO\_UDP\_SET\_SERVICE See usage for [globusxio\\_attr.cntl](#).  
 GLOBUS\_XIO\_UDP\_GET\_SERVICE See usage for [globusxio\\_attr.cntl](#).  
 GLOBUS\_XIO\_UDP\_SET\_PORT See usage for [globusxio\\_attr.cntl](#).  
 GLOBUS\_XIO\_UDP\_GET\_PORT See usage for [globusxio\\_attr.cntl](#).  
 GLOBUS\_XIO\_UDP\_SET\_LISTEN\_RANGE See usage for [globusxio\\_attr.cntl](#).  
 GLOBUS\_XIO\_UDP\_GET\_LISTEN\_RANGE See usage for [globusxio\\_attr.cntl](#).  
 GLOBUS\_XIO\_UDP\_SET\_INTERFACE See usage for [globusxio\\_attr.cntl](#).  
 GLOBUS\_XIO\_UDP\_GET\_INTERFACE See usage for [globusxio\\_attr.cntl](#).  
 GLOBUS\_XIO\_UDP\_SET\_RESTRICT\_PORT See usage for [globusxio\\_attr.cntl](#).  
 GLOBUS\_XIO\_UDP\_GET\_RESTRICT\_PORT See usage for [globusxio\\_attr.cntl](#).  
 GLOBUS\_XIO\_UDP\_SET\_REUSEADDR See usage for [globusxio\\_attr.cntl](#).  
 GLOBUS\_XIO\_UDP\_GET\_REUSEADDR See usage for [globusxio\\_attr.cntl](#).  
 GLOBUS\_XIO\_UDP\_SET\_NO\_IPV6 See usage for [globusxio\\_attr.cntl](#).  
 GLOBUS\_XIO\_UDP\_GET\_NO\_IPV6 See usage for [globusxio\\_attr.cntl](#).  
 GLOBUS\_XIO\_UDP\_GET\_HANDLE See usage for [globusxio\\_attr.cntl](#), [globusxio\\_handlecntl](#).  
 GLOBUS\_XIO\_UDP\_SET\_SNDBUF See usage for [globusxio\\_attr.cntl](#), [globusxio\\_handlecntl](#).  
 GLOBUS\_XIO\_UDP\_GET\_SNDBUF See usage for [globusxio\\_attr.cntl](#), [globusxio\\_handlecntl](#).  
 GLOBUS\_XIO\_UDP\_SET\_RCVBUF See usage for [globusxio\\_attr.cntl](#), [globusxio\\_handlecntl](#).  
 GLOBUS\_XIO\_UDP\_GET\_RCVBUF See usage for [globusxio\\_attr.cntl](#), [globusxio\\_handlecntl](#).  
 GLOBUS\_XIO\_UDP\_GET\_CONTACT See usage for [globusxio\\_handlecntl](#), [globusxio\\_datadescriptor-cntl](#).  
 GLOBUS\_XIO\_UDP\_GET\_NUMERIC\_CONTACT See usage for: [globusxio\\_handlecntl](#), [globusxio\\_datadescriptorcntl](#).  
 GLOBUS\_XIO\_UDP\_SET\_CONTACT See usage for [globusxio\\_datadescriptorcntl](#).  
 GLOBUS\_XIO\_UDP\_CONNECT See usage for [globusxio\\_handlecntl](#).  
 GLOBUS\_XIO\_UDP\_SET\_MULTICAST See usage for [globusxio\\_attr.cntl](#).

## 6.46.3 Function Documentation

6.46.3.1 `globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_SET_HANDLE, globus_xio_system_socket_t handle)`

Set the udp socket to use.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`handle` Use this handle (fd or SOCKET). Note: `close()` will not be called on this handle.

6.46.3.2 `globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_SET_SERVICE, const char servicename)`

Set the udp service name to listen on.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`servicename` The service name to use when setting up the listener. If the service name cannot be resolved, the port (if one is set) will be used instead.

6.46.3.3 `globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_GET_SERVICE, char service_name_out)`

Get the service name to listen on.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`service_name_out` A pointer to the service name will be stored here. If none is set, NULL will be passed back. Otherwise, the name will be duplicated with `strdup()` and the user should call `free()` on it.

6.46.3.4 `globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_SET_PORT, int listener_port)`

Set the port number to listen on.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

The default is 0 (system assigned)

Parameters:

`listener_port` The port number to use when setting up the listener. If the service name is also set, this will only be used if that can't be resolved.

6.46.3.5 `globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_GET_PORT, int listener_port_out)`

the port number to listen on.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`listener_port_out` The port will be stored here.

6.46.3.6 `globusresult_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_SET_LISTEN_RANGE, int listener_min_port, int listener_max_port)`

Set the port range to connect the listener to.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used only where no specific service or port has been set. It overrides the range set in the `GLOBUS_PORT_RANGE` env variable. If 'restrict port' is true, the listening port will be constrained to the range specified.

Parameters:

`listener_min_port` The lower bound on the listener port. (default 0 – no bound)

`listener_max_port` The upper bound on the listener port. (default 0 – no bound)

See also:

[GLOBUS\\_XIO\\_UDP\\_SET\\_RESTRICT\\_PORT](#)

6.46.3.7 `globusresult_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_GET_LISTEN_RANGE, int listener_min_port_out, int listener_max_port_out)`

Get the udp port range on an attr.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`listener_min_port_out` The lower bound will be stored here.

`listener_max_port_out` The upper bound will be stored here.

6.46.3.8 `globusresult_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_SET_RESTRICT_PORT, globus_bool_t restrict_port)`

Enable or disable the listener range constraints.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This enables or ignores the port range found in the attr or in the env. By default, those ranges are enabled.

Parameters:

`restrict_port` `GLOBUS_TRUE` to enable (default), `GLOBUS_FALSE` to disable.

See also:

[GLOBUS\\_XIO\\_UDP\\_SET\\_LISTEN\\_RANGE](#)

6.46.3.9 `globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_GET_RESTRICT_PORT, globus_bool_t restrict_port_out)`

Get the restrict port `ag`.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`restrict_port_out` The restrict port `ag` will be stored here.

6.46.3.10 `globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_GET_HANDLE, globus_xio_system_socket_t handle_out)`

Get the socket handle on an `attr` or `handle`.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`handle_out` The udp socket will be stored here. If none is set, `GLOBUS_XIO_UDP_INVALID_HANDLE` will be set.

6.46.3.11 `globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_UDP_GET_HANDLE, globus_xio_system_socket_t handle_out)`

Get the socket handle on an `attr` or `handle`.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

`handle_out` The udp socket will be stored here. If none is set, `GLOBUS_XIO_UDP_INVALID_HANDLE` will be set.

6.46.3.12 `globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_UDP_SET_SNDBUF, int sndbuf)`

Set the socket send buffer size.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used to set the size of the send buffer used on the socket.

Parameters:

`sndbuf` The send buffer size in bytes to use. (default is system specific)

6.46.3.13 `globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_UDP_GET_SNDBUF, int sndbuf_out)`

Get the send buffer size on the `attr` or `handle`.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

## Parameters:

sndbuf\_out The send buffer size will be stored here.

6.46.3.14 `globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_UDP_GET_CONTACT, char contactstring_out)`

Get the contact string associated with a handle or data descriptor.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Use with [globus\\_xio\\_handle\\_cntl\(\)](#) to get a contact string for the udp listener. Use with [globus\\_xio\\_data\\_descriptor\\_cntl\(\)](#) to get the sender's contact string from a data descriptor passed to [globus\\_xio\\_registerread\(\)](#)

## Parameters:

contactstring\_out A pointer to a contact string will be stored here. The user should free() it when done with it. It will be in the format: <hostname>:<port>

## See also:

[GLOBUS\\_XIO\\_GET\\_LOCAL\\_CONTACT](#)

6.46.3.15 `globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_UDP_GET_CONTACT, char contactstring_out)`

Get the contact string associated with a handle or data descriptor.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Use with [globus\\_xio\\_handle\\_cntl\(\)](#) to get a contact string for the udp listener. Use with [globus\\_xio\\_data\\_descriptor\\_cntl\(\)](#) to get the sender's contact string from a data descriptor passed to [globus\\_xio\\_registerread\(\)](#)

## Parameters:

contactstring\_out A pointer to a contact string will be stored here. The user should free() it when done with it. It will be in the format: <hostname>:<port>

## See also:

[GLOBUS\\_XIO\\_GET\\_LOCAL\\_CONTACT](#)

6.46.3.16 `globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_UDP_SET_CONTACT, char contactstring)`

Set the destination contact.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Use on a data descriptor passed to [globus\\_xio\\_registerwrite\(\)](#) to specify the recipient of the data. This is necessary with unconnected handles or to send to recipients other than the connected one.

## Parameters:

contactstring A pointer to a contact string of the format: <hostname/ip>:<port/service>

## See also:

[GLOBUS\\_XIO\\_UDP\\_CONNECT](#)

6.46.3.17 `globus_result_t globus_xio_handle_cntl` (handle, driver, GLOBUS\_XIO\_UDP\_CONNECT, char contact.string)

Set the default destination contact.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Connecting a handle to a specific contact blocks packets from any other contact. It also sets the default destination of all outgoing packets so, using [Attributes and Cntls](#) is unnecessary.

Parameters:

contact.string A pointer to a contact string of the format `hostname/ip[:<port/service>`

6.46.3.18 `globus_result_t globus_xio_attr_cntl` (attr, driver, GLOBUS\_XIO\_UDP\_SET\_MULTICAST, char contact.string)

Join a multicast group.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Specify a multicast group to join. All packets received will be to the specified multicast address. Do not use [Attributes and Cntls](#) or pass a contact string on the open. Consider using [Attributes and Cntls](#) to allow other apps to join this group. Use [Attributes and Cntls](#) to specify the interface to use. Will not affect handles set with [Attributes and Cntls](#). [Attributes and Cntls](#) is ignored.

Parameters:

contact.string A pointer to a contact string of the multicast group to join with the format: `<hostname/ip[:<port/service>`

## 6.47 Types

Defines

`#define GLOBUS_XIO_UDP_INVALID_HANDLE`

### 6.47.1 Define Documentation

#### 6.47.1.1 `#define GLOBUS_XIO_UDP_INVALID_HANDLE`

Invalid handle type.

See also:

[GLOBUS\\_XIO\\_UDP\\_SET\\_HANDLE](#)

## 6.48 Error Types

Enumerations

`enum globus_xio_udp_error_type_t` if `GLOBUS_XIO_UDP_ERROR_NO_ADDRS`, `GLOBUS_XIO_UDP_ERROR_SHORT_WRITE`

### 6.48.1 Detailed Description

The UDP driver is very close to the system code, so most errors reported by it are converted from the system `errno`. A few of the exceptions are `GLOBUS_XIO_ERROR_COMMAND`, `GLOBUS_XIO_ERROR_CONTACT_STRING`, `GLOBUS_XIO_ERROR_CANCELED`, [Error Types](#) and [Error Types](#)

See also:

[globusxio\\_driver.error.match\(\)](#), `globuserror.errno.match()`

### 6.48.2 Enumeration Type Documentation

#### 6.48.2.1 enum globusxio\_udp\_error\_type\_t

UDP driver specific error types.

Enumeration values:

`GLOBUS_XIO_UDP_ERROR_NO_ADDRS` Indicates that no IPv4/6 compatible sockets could be resolved for the specified hostname.

`GLOBUS_XIO_UDP_ERROR_SHORT_WRITE` Indicates that a write of the full buffer failed. Possibly need to increase the send buffer size.

## 7 globus xio Data Structure Documentation

### 7.1 globusxio\_http\_header\_t Struct Reference

HTTP Header.

Data Fields

```
char name
char value
```

#### 7.1.1 Detailed Description

HTTP Header.

#### 7.1.2 Field Documentation

##### 7.1.2.1 char globusxio\_http\_header\_t::name

Header Name.

##### 7.1.2.2 char globusxio\_http\_header\_t::value

Header Value.

## 8 globus xio File Documentation

### 8.1 globusxio\_ le \_driver.h File Reference

Header le for XIO File Driver.

De nes

```
#de ne GLOBUS_XIO_FILE_INVALID_HANDLE
```

Enumerations

```
enum globusxio_ le _attr.cmd.t f GLOBUS_XIO_FILE_SET_MODE, GLOBUS_XIO_FILE_GET_MODE,
GLOBUS_XIO_FILE_SET_FLAGS, GLOBUS_XIO_FILE_GET_FLAGS, GLOBUS_XIO_FILE_SET_
TRUNC_OFFSET, GLOBUS_XIO_FILE_GET_TRUNC_OFFSET, GLOBUS_XIO_FILE_SET_HANDLE,
GLOBUS_XIO_FILE_GET_HANDLE, GLOBUS_XIO_FILE_SET_BLOCKING_IO, GLOBUS_XIO_FILE_
GET_BLOCKING_IO, GLOBUS_XIO_FILE_SEEK.g
enum globusxio_ le _ag.t f GLOBUS_XIO_FILE_CREAT = O_CREAT, GLOBUS_XIO_FILE_EXCL =
O_EXCL, GLOBUS_XIO_FILE_RDONLY = O_RDONLY, GLOBUS_XIO_FILE_WRONLY = O_WRONLY,
GLOBUS_XIO_FILE_RDWR = O_RDWR, GLOBUS_XIO_FILE_TRUNC = O_TRUNC, GLOBUS_XIO_
FILE_APPEND = O_APPEND, GLOBUS_XIO_FILE_BINARY = 0, GLOBUS_XIO_FILE_TEXT = 0.g
enum globusxio_ le _modet f GLOBUS_XIO_FILE_IRWXU = S_IRWXU, GLOBUS_XIO_FILE_IRUSR =
S_IRUSR, GLOBUS_XIO_FILE_IWUSR = S_IWUSR, GLOBUS_XIO_FILE_IXUSR = S_IXUSR, GLOBUS_
XIO_FILE_IRWXO = S_IRWXO, GLOBUS_XIO_FILE_IROTH = S_IROTH, GLOBUS_XIO_FILE_IWOTH =
S_IWOTH, GLOBUS_XIO_FILE_IXOTH = S_IXOTH, GLOBUS_XIO_FILE_IRWXG = S_IRWXG,
GLOBUS_XIO_FILE_IRGRP = S_IRGRP, GLOBUS_XIO_FILE_IWGRP = S_IWGRP, GLOBUS_XIO_FILE_
IXGRP = S_IXGRP.g
enum globusxio_ le _whencet f GLOBUS_XIO_FILE_SEEK_SET = SEEK_SET, GLOBUS_XIO_FILE_
SEEK_CUR = SEEK_CUR, GLOBUS_XIO_FILE_SEEK_END = SEEK_END.g
```

Functions

```
globusresultt globusxio_attr.cntl (attr, driver, GLOBUS_XIO_FILE_SET_MODE, int mode)
globusresultt globusxio_attr.cntl (attr, driver, GLOBUS_XIO_FILE_GET_MODE, int modeout)
globusresultt globusxio_attr.cntl (attr, driver, GLOBUS_XIO_FILE_SET_TRUNC_OFFSET, globusoff_t off-
set)
globusresultt globusxio_attr.cntl (attr, driver, GLOBUS_XIO_FILE_GET_TRUNC_OFFSET, globusoff_t
offsetout)
globusresultt globusxio_attr.cntl (attr, driver, GLOBUS_XIO_FILE_SET_HANDLE, globusxio_system le _t
handle)
globusresultt globusxio_attr.cntl (attr, driver, GLOBUS_XIO_FILE_GET_HANDLE, globusxio_system-
le _t handleout)
globusresultt globusxio_handlecntl (handle, driver, GLOBUS_XIO_FILE_GET_HANDLE, globusxio_-
system le _t handleout)
globusresultt globusxio_attr.cntl (attr, driver, GLOBUS_XIO_FILE_SET_BLOCKING_IO, globusbool-
t useblocking.io)
globusresultt globusxio_handlecntl (handle, driver, GLOBUS_XIO_FILE_SET_BLOCKING_IO, globus-
boolt useblocking.io)
globusresultt globusxio_attr.cntl (attr, driver, GLOBUS_XIO_FILE_GET_BLOCKING_IO, globusbool-
t useblocking.io.out)
```



globusresultt [globusxio\\_handlecntl](#) (handle, driver, GLOBUSXIO\_FILE\_GET\_BLOCKING\_IO, globus-boolt useblockingio\_out)  
 globusresultt [globusxio\\_handlecntl](#) (handle, driver, GLOBUSXIO\_FILE\_SEEK, globusoff\_t in\_out\_offset, [globusxio\\_le\\_whencet](#) whence)

### 8.1.1 Detailed Description

Header file for XIO File Driver.

## 8.2 globusxio\_http.h File Reference

Globus XIO HTTP Driver Header.

### Data Structures

struct [globusxio\\_http\\_header\\_t](#)  
 HTTP Header.

### Enumerations

enum [globusxio\\_http\\_handlecmd\\_t](#) f GLOBUSXIO\_HTTP\_HANDLE\_SET\_RESPONSEHEADER, GLOBUSXIO\_HTTP\_HANDLE\_SET\_RESPONSESTATUSCODE, GLOBUSXIO\_HTTP\_HANDLE\_SET\_RESPONSEREASONPHRASE, GLOBUSXIO\_HTTP\_HANDLE\_SET\_RESPONSEHTTP\_VERSION, GLOBUSXIO\_HTTP\_HANDLE\_SET\_END\_OF\_ENTITY g  
 enum [globusxio\\_http\\_attr\\_cmd\\_t](#) f GLOBUSXIO\_HTTP\_ATTR\_SET\_REQUESTMETHOD, GLOBUSXIO\_HTTP\_ATTR\_SET\_REQUESTHTTP\_VERSION, GLOBUSXIO\_HTTP\_ATTR\_SET\_REQUESTHEADER, GLOBUSXIO\_HTTP\_ATTR\_DELAY\_WRITE\_HEADER, GLOBUSXIO\_HTTP\_GET\_REQUEST, GLOBUSXIO\_HTTP\_GET\_RESPONSE g  
 enum [globusxio\\_http\\_errorst](#) f GLOBUSXIO\_HTTP\_ERROR\_INVALID\_HEADER, GLOBUSXIO\_HTTP\_ERROR\_PARSE, GLOBUSXIO\_HTTP\_ERROR\_NO\_ENTITY, GLOBUSXIO\_HTTP\_ERROR\_EOF, GLOBUSXIO\_HTTP\_ERROR\_PERSISTENT\_CONNECTION\_DROPPED g  
 enum [globusxio\\_http\\_version\\_t](#) f , GLOBUSXIO\_HTTP\_VERSION\_1\_0, GLOBUSXIO\_HTTP\_VERSION\_1\_1 g

### Functions

globusresultt [globusxio\\_handlecntl](#) (handle, driver, GLOBUSXIO\_HTTP\_HANDLE\_SET\_RESPONSEHEADER, const char headename, const char headervalue)  
 globusresultt [globusxio\\_handlecntl](#) (handle, driver, GLOBUSXIO\_HTTP\_HANDLE\_SET\_RESPONSESTATUSCODE, int status)  
 globusresultt [globusxio\\_handlecntl](#) (handle, driver, GLOBUSXIO\_HTTP\_HANDLE\_SET\_RESPONSEREASONPHRASE, const char reason)  
 globusresultt [globusxio\\_handlecntl](#) (handle, driver, GLOBUSXIO\_HTTP\_HANDLE\_SET\_RESPONSEHTTP\_VERSION, [globusxio\\_http\\_version\\_t](#) version)  
 globusresultt [globusxio\\_handlecntl](#) (handle, driver, GLOBUSXIO\_HTTP\_HANDLE\_SET\_END\_OF\_ENTITY)  
 globusresultt [globusxio\\_attr\\_cntl](#) (attr, driver, GLOBUSXIO\_HTTP\_ATTR\_SET\_REQUESTMETHOD, const char method)

```

globusresultt globusxio\_attr\_cntl (attr, driver, GLOBUSXIO_HTTP_ATTR_SET.REQUESTHTTP-
VERSION,globusxio\_http\_versiont version)
globusresultt globusxio\_attr\_cntl (attr, driver, GLOBUSXIO_HTTP_ATTR_SET.REQUESTHEADER,
const char headename, const charheadervalue)
globusresultt globusxio\_attr\_cntl (attr, driver, GLOBUSXIO_HTTP_ATTR_DELAY_WRITE_HEADER)
globusresultt globusxio\_data\_descriptor\_cntl (dd, driver, GLOBUSXIO_HTTP_GET_REQUEST, char
method, char uri, globusxio\_http\_versiont http_version, globus_hashtablet headers)
globusresultt globusxio\_data\_descriptor\_cntl (dd, driver, GLOBUSXIO_HTTP_GET_RESPONSE, int
statuscode, char reasonphraseglobusxio\_http\_versiont http_version, globus_hashtablet headers)

```

### 8.2.1 Detailed Description

Globus XIO HTTP Driver Header.

### 8.2.2 Function Documentation

8.2.2.1 `globusresult_t globus_xio_attr_cntl` (attr, driver, GLOBUS\_XIO\_HTTP\_ATTR\_DELAY\_WRITE\_HEADER)

Delay writing HTTP request until rst data write.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

If this attribute is present when opening an HTTP handle, the HTTP request will not be sent immediately upon opening the handle. Instead, it will be delayed until the rst data write is done. This allows other HTTP headers to be sent after the handle is opened.

This attribute cntl takes no arguments.

8.2.2.2 `globusresult_t globus_xio_data_descriptor_cntl` (dd, driver, GLOBUS\_XIO\_HTTP\_GET\_REQUEST, char method, char uri, [globusxio\\_http\\_versiont](#) http\_version, globus\_hashtablet headers)

Get HTTP Request Information.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Returns in the passed parameters values concerning the HTTP request. Any of the parameters may be NULL if the application is not interested in that part of the information.

Parameters:

method Pointer to be set to the HTTP request method (typically GET, PUT, or POST). The caller must not access this value outside of the lifetime of the data descriptor nor free it.

uri Pointer to be set to the requested HTTP path. The caller must not access this value outside of the lifetime of the data descriptor nor free it.

http\_version Pointer to be set to the HTTP version used for this request.

headers Pointer to be set to point to a hashtable of [globusxio\\_http\\_header](#) values, keyed by the HTTP header names. The caller must not access this value outside of the lifetime of the data descriptor nor free it or any values in it.

8.2.2.3 globusresult\_t globus\_xio\_data\_descriptor\_cntl (dd, driver, GLOBUS\_XIO\_HTTP\_GET\_RESPONSE, int statuscode, char reasonphrase, globus\_xio\_http\_version\_t http\_version, globus\_hashtable\_t headers)

Get HTTP Response Information.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Returns in the passed parameters values concerning the HTTP response. Any of the parameters may be NULL if the application is not interested in that part of the information.

Parameters:

statuscode Pointer to be set to the HTTP response status code (such as 404), as per RFC 2616. The caller must not access this value outside of the lifetime of the data descriptor nor free it or any values in it.

reasonphrase Pointer to be set to the HTTP response reason phrase (such as Not Found). The caller must not access this value outside of the lifetime of the data descriptor nor free it or any values in it.

http\_version Pointer to be set to the HTTP version used for this request.

headers Pointer to be set to point to a hashtable of globus\_xio\_http\_header\_t values, keyed by the HTTP header names. The caller must not access this value outside of the lifetime of the data descriptor nor free it or any values in it.

## 8.3 globusxio\_mode.e.driver.h File Reference

Header file for XIO MODE.E Driver.

Enumerations

```
enum globusxio_mode.e.error.type_t { GLOBUS_XIO_MODE.E.HEADER_ERROR,
enum globusxio_mode.e.cmd_t { GLOBUS_XIO_MODE.E.SET_STACK, GLOBUS_XIO_MODE.E.GET_STACK,
GLOBUS_XIO_MODE.E.SET_NUM_STREAMS, GLOBUS_XIO_MODE.E.GET_NUM_STREAMS,
GLOBUS_XIO_MODE.E.SET_OFFSET_READS, GLOBUS_XIO_MODE.E.GET_OFFSET_READS,
GLOBUS_XIO_MODE.E.SET_MANUAL_EODC, GLOBUS_XIO_MODE.E.GET_MANUAL_EODC,
GLOBUS_XIO_MODE.E.SEND_EOD, GLOBUS_XIO_MODE.E.SET_EODC, GLOBUS_XIO_MODE.E.DD.GET.OFFSET,
GLOBUS_XIO_MODE.E.SET_STACK_ATTR, GLOBUS_XIO_MODE.E.GET_STACK_ATTR } g
```

Functions

```
globusresult_t globusxio_attr_cntl (attr, driver, GLOBUSXIO_MODE.E.SET_STACK, globusxio_stack_t stack)
globusresult_t globusxio_attr_cntl (attr, driver, GLOBUSXIO_MODE.E.GET_STACK, globusxio_stack_t stackout)
globusresult_t globusxio_attr_cntl (attr, driver, GLOBUSXIO_MODE.E.SET_NUM_STREAMS, int num_streams)
globusresult_t globusxio_attr_cntl (attr, driver, GLOBUSXIO_MODE.E.GET_NUM_STREAMS, int num_streamsout)
globusresult_t globusxio_attr_cntl (attr, driver, GLOBUSXIO_MODE.E.SET_OFFSET_READS, globus_bool_t offset_reads)
globusresult_t globusxio_attr_cntl (attr, driver, GLOBUSXIO_MODE.E.GET_OFFSET_READS, globus_bool_t offset_readsout)
```

```

globus_result_t globusxio_data_descriptorctl (dd, driver, GLOBUSXIO_MODE_E_SENDEOD, globus-
bool_t sendeod)
globus_result_t globusxio_handlectl (handle, driver, GLOBUSXIO_MODE_E_SETEODC, int eodcount)
globus_result_t globusxio_data_descriptorctl (dd, driver, GLOBUSXIO_MODE_E_DD_GET_OFFSET,
globus_off_t offset_out)
globus_result_t globusxio_attrctl (attr, driver, GLOBUSXIO_MODE_E_GET_STACK_ATTR, globusxio-
attr_t stack_out)

```

### 8.3.1 Detailed Description

Header file for XIO MODE.E Driver.

## 8.4 globusxio\_ordering\_driver.h File Reference

Header file for XIO ORDERING Driver.

### Enumerations

```

enum globusxio_ordering_error_type_t { GLOBUSXIO_ORDERING_ERROR_READ, GLOBUSXIO_-
ORDERING_ERROR_CANCEL }
enum globusxio_ordering_cmd_t { GLOBUSXIO_ORDERING_SET_OFFSET, GLOBUSXIO_-
ORDERING_SET_MAX_READ_COUNT, GLOBUSXIO_ORDERING_GET_MAX_READ_COUNT,
GLOBUSXIO_ORDERING_SET_BUFFERING, GLOBUSXIO_ORDERING_GET_BUFFERING,
GLOBUSXIO_ORDERING_SET_BUF_SIZE, GLOBUSXIO_ORDERING_GET_BUF_SIZE, GLOBUS-
XIO_ORDERING_SET_MAX_BUF_COUNT, GLOBUSXIO_ORDERING_GET_MAX_BUF_COUNT }

```

### Functions

```

globus_result_t globusxio_handlectl (handle, driver, GLOBUSXIO_ORDERING_SET_OFFSET, globus-
off_t offset)
globus_result_t globusxio_attrctl (attr, driver, GLOBUSXIO_ORDERING_SET_MAX_READ_COUNT, int
max_read_count)
globus_result_t globusxio_attrctl (attr, driver, GLOBUSXIO_ORDERING_GET_MAX_READ_COUNT, int
max_read_count_out)
globus_result_t globusxio_attrctl (attr, driver, GLOBUSXIO_ORDERING_SET_BUFFERING, globus-
bool_t buffering)
globus_result_t globusxio_attrctl (attr, driver, GLOBUSXIO_ORDERING_GET_BUFFERING, globus-
bool_t buffering_out)

```

### 8.4.1 Detailed Description

Header file for XIO ORDERING Driver.

## 8.5 globusxio\_tcp\_driver.h File Reference

Header file for XIO TCP Driver.

De nes

```
#define GLOBUS_XIO_TCP_INVALID_HANDLE
```

Enumerations

```
enum globusxio_tcp_error_type_t { GLOBUS_XIO_TCP_ERROR_NO_ADDRS,
enum globusxio_tcp_cmd_t { GLOBUS_XIO_TCP_SET_SERVICE, GLOBUS_XIO_TCP_GET_SERVICE,
GLOBUS_XIO_TCP_SET_PORT, GLOBUS_XIO_TCP_GET_PORT, GLOBUS_XIO_TCP_SET_BACKLOG,
GLOBUS_XIO_TCP_GET_BACKLOG, GLOBUS_XIO_TCP_SET_LISTEN_RANGE, GLOBUS_XIO_TCP_GET_LISTEN_RANGE,
GLOBUS_XIO_TCP_GET_HANDLE, GLOBUS_XIO_TCP_SET_HANDLE,
GLOBUS_XIO_TCP_SET_INTERFACE, GLOBUS_XIO_TCP_GET_INTERFACE, GLOBUS_XIO_TCP_SET_RESTRICT_PORT,
GLOBUS_XIO_TCP_GET_RESTRICT_PORT, GLOBUS_XIO_TCP_SET_REUSEADDR, GLOBUS_XIO_TCP_GET_REUSEADDR,
GLOBUS_XIO_TCP_SET_NO_IPV6, GLOBUS_XIO_TCP_GET_NO_IPV6, GLOBUS_XIO_TCP_SET_CONNECT_RANGE,
GLOBUS_XIO_TCP_GET_CONNECT_RANGE, GLOBUS_XIO_TCP_SET_KEEPALIVE, GLOBUS_XIO_TCP_GET_KEEPALIVE,
GLOBUS_XIO_TCP_SET_LINGER, GLOBUS_XIO_TCP_GET_LINGER, GLOBUS_XIO_TCP_SET_OOBINLINE,
GLOBUS_XIO_TCP_GET_OOBINLINE, GLOBUS_XIO_TCP_SET_SNDBUF, GLOBUS_XIO_TCP_GET_SNDBUF,
GLOBUS_XIO_TCP_SET_RCVBUF, GLOBUS_XIO_TCP_GET_RCVBUF, GLOBUS_XIO_TCP_SET_NODELAY,
GLOBUS_XIO_TCP_GET_NODELAY, GLOBUS_XIO_TCP_SET_SEND_FLAGS, GLOBUS_XIO_TCP_GET_SEND_FLAGS,
GLOBUS_XIO_TCP_GET_LOCAL_CONTACT, GLOBUS_XIO_TCP_GET_LOCAL_NUMERIC_CONTACT,
GLOBUS_XIO_TCP_GET_REMOTE_CONTACT, GLOBUS_XIO_TCP_GET_REMOTE_NUMERIC_CONTACT,
GLOBUS_XIO_TCP_AFFECT_ATTR_DEFAULTS, GLOBUS_XIO_TCP_SET_BLOCKING_IO, GLOBUS_XIO_TCP_GET_BLOCKING_IO
enum globusxio_tcp_send_flags_t { GLOBUS_XIO_TCP_SEND_OOB = MSG_OOB }
```

Functions

```
globus_result_t globusxio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_SERVICE, const char service_name)
globus_result_t globusxio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_SERVICE, char service_name_out)
globus_result_t globusxio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_PORT, int listener_port)
globus_result_t globusxio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_PORT, int listener_port_out)
globus_result_t globusxio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_LISTEN_RANGE, int listener_min_port, int listener_max_port)
globus_result_t globusxio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_LISTEN_RANGE, int listener_min_port_out, int listener_max_port_out)
globus_result_t globusxio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_HANDLE, globusxio_system_socket_t handle_out)
globus_result_t globusxio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_GET_HANDLE, globusxio_system_socket_t handle_out)
globus_result_t globusxio_server_cntl (server, driver, GLOBUS_XIO_TCP_GET_HANDLE, globusxio_system_socket_t handle_out)
globus_result_t globusxio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_HANDLE, globusxio_system_socket_t handle)
globus_result_t globusxio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_RESTRICT_PORT, globus_bool_t restrict_port)
globus_result_t globusxio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_RESTRICT_PORT, globus_bool_t restrict_port_out)
```

```

globusresultt globusxio\_handlecntl (handle, driver, GLOBUSXIO_TCP_SET_KEEPALIVE, globusboolt
keepalive)
globusresultt globusxio\_handlecntl (handle, driver, GLOBUSXIO_TCP_GET_KEEPALIVE, globusboolt
keepaliveout)
globusresultt globusxio\_attrcntl (attr, driver, GLOBUSXIO_TCP_SET_LINGER, globusboolt linger, int
linger.time)
globusresultt globusxio\_handlecntl (handle, driver, GLOBUSXIO_TCP_SET_LINGER, globusbool-
t linger, int linger.time)
globusresultt globusxio\_attrcntl (attr, driver, GLOBUSXIO_TCP_GET_LINGER, globusboolt linger.out,
int linger.time.out)
globusresultt globusxio\_handlecntl (handle, driver, GLOBUSXIO_TCP_GET_LINGER, globusbool-
t linger.out, int linger.time.out)
globusresultt globusxio\_handlecntl (handle, driver, GLOBUSXIO_TCP_SET_SNDBUF, int sndbuf)
globusresultt globusxio\_handlecntl (handle, driver, GLOBUSXIO_TCP_GET_SNDBUF, int sndbufout)
globusresultt globusxio\_datadescriptorcntl (dd, driver, GLOBUSXIO_TCP_SET_SEND_FLAGS, int
send.ags)
globusresultt globusxio\_datadescriptorcntl (dd, driver, GLOBUSXIO_TCP_GET_SEND_FLAGS, int
send.ags.out)
globusresultt globusxio\_handlecntl (handle, driver, GLOBUSXIO_TCP_GET_LOCAL_CONTACT, char
contactstring.out)
globusresultt globusxio\_servercntl (server, driver, GLOBUSXIO_TCP_GET_LOCAL_CONTACT, char
contactstring.out)

```

### 8.5.1 Detailed Description

Header file for XIO TCP Driver.

## 8.6 globusxio\_udp\_driver.h File Reference

Header file for XIO UDP Driver.

Defines

```
#define GLOBUS\_XIO\_UDP\_INVALID\_HANDLE
```

Enumerations

```

enum globusxio\_udp\_error\_type\_t { GLOBUS\_XIO\_UDP\_ERROR\_NO\_ADDRS, GLOBUS\_XIO\_UDP-
ERROR\_SHORT\_WRITE }
enum globusxio\_udp\_cmd\_t { GLOBUS\_XIO\_UDP\_SET\_HANDLE, GLOBUS\_XIO\_UDP\_SET\_SERVICE,
GLOBUS\_XIO\_UDP\_GET\_SERVICE, GLOBUS\_XIO\_UDP\_SET\_PORT, GLOBUS\_XIO\_UDP\_GET\_PORT,
GLOBUS\_XIO\_UDP\_SET\_LISTEN\_RANGE, GLOBUS\_XIO\_UDP\_GET\_LISTEN\_RANGE, GLOBUS\_XIO-
UDP\_SET\_INTERFACE, GLOBUS\_XIO\_UDP\_GET\_INTERFACE, GLOBUS\_XIO\_UDP\_SET\_RESTRICT-
PORT, GLOBUS\_XIO\_UDP\_GET\_RESTRICTPORT, GLOBUS\_XIO\_UDP\_SET\_REUSEADDR, GLOBUS-
XIO\_UDP\_GET\_REUSEADDR, GLOBUS\_XIO\_UDP\_SET\_NO\_IPV6, GLOBUS\_XIO\_UDP\_GET\_NO\_IPV6,
GLOBUS\_XIO\_UDP\_GET\_HANDLE, GLOBUS\_XIO\_UDP\_SET\_SNDBUF, GLOBUS\_XIO\_UDP\_GET-
SNDBUF, GLOBUS\_XIO\_UDP\_SET\_RCVBUF, GLOBUS\_XIO\_UDP\_GET\_RCVBUF, GLOBUS\_XIO-
UDP\_GET\_CONTACT, GLOBUS\_XIO\_UDP\_GET\_NUMERIC\_CONTACT, GLOBUS\_XIO\_UDP\_SET-
CONTACT, GLOBUS\_XIO\_UDP\_CONNECT, GLOBUS\_XIO\_UDP\_SET\_MULTICAST }

```

## Functions

```

globusresultt globusxio\_attr\_cntl (attr, driver, GLOBUSXIO_UDP_SET_HANDLE, globusxio_system-
socket handle)
globusresultt globusxio\_attr\_cntl (attr, driver, GLOBUSXIO_UDP_SET_SERVICE, const char service-
name)
globusresultt globusxio\_attr\_cntl (attr, driver, GLOBUSXIO_UDP_GET_SERVICE, char servicename-
out)
globusresultt globusxio\_attr\_cntl (attr, driver, GLOBUSXIO_UDP_SET_PORT, int listenerport)
globusresultt globusxio\_attr\_cntl (attr, driver, GLOBUSXIO_UDP_GET_PORT, int listenerportout)
globusresultt globusxio\_attr\_cntl (attr, driver, GLOBUSXIO_UDP_SET_LISTEN_RANGE, int listenermin-
port, int listenermaxport)
globusresultt globusxio\_attr\_cntl (attr, driver, GLOBUSXIO_UDP_GET_LISTEN_RANGE, int listener-
minportout, int listenermaxportout)
globusresultt globusxio\_attr\_cntl (attr, driver, GLOBUSXIO_UDP_SET_RESTRICTPORT, globusboolt
restrictport)
globusresultt globusxio\_attr\_cntl (attr, driver, GLOBUSXIO_UDP_GET_RESTRICTPORT, globusboolt
restrictportout)
globusresultt globusxio\_attr\_cntl (attr, driver, GLOBUSXIO_UDP_GET_HANDLE, globusxio_system-
socket handleout)
globusresultt globusxio\_handlecntl (handle, driver, GLOBUSXIO_UDP_GET_HANDLE, globusxio_-
systemsocket handleout)
globusresultt globusxio\_handlecntl (handle, driver, GLOBUSXIO_UDP_SET_SNDBUF, int sndbuf)
globusresultt globusxio\_handlecntl (handle, driver, GLOBUSXIO_UDP_GET_SNDBUF, int sndbufout)
globusresultt globusxio\_handlecntl (handle, driver, GLOBUSXIO_UDP_GET_CONTACT, char contact-
stringout)
globusresultt globusxio\_datadescriptorcntl (dd, driver, GLOBUSXIO_UDP_GET_CONTACT, char
contactstringout)
globusresultt globusxio\_datadescriptorcntl (dd, driver, GLOBUSXIO_UDP_SET_CONTACT, char
contactstring)
globusresultt globusxio\_handlecntl (handle, driver, GLOBUSXIO_UDP_CONNECT, char contactstring)
globusresultt globusxio\_attr\_cntl (attr, driver, GLOBUSXIO_UDP_SET_MULTICAST, char contactstring)

```

### 8.6.1 Detailed Description

Header file for XIO UDP Driver.

## 9 globus xio Page Documentation

### 9.1 Data descriptors

globusxio uses data descriptors to associate meta data with the data being written or the data read.

Data descriptors flow into the drivers read and write interface functions by way of the operation structure. If the driver is interested in viewing the data descriptor it can request it from the operation structure via a call to `globus_driver_operation_get_data_descriptor()` and it can view any driver specific data descriptor via a call to `globus_driver_data_descriptor_get_specic()`. The driver can modify values in the data descriptor by setting values before passing the request down the stack. Several functions are available to modify the data descriptors. There is no need to "set()" the data descriptors back into the operation. The functions for manipulating the values in a DD affect the values xio has directly.



Data descriptors flow back to the driver in the callbacks for the data operations. When calling finished operation on a data operation the driver must pass in a data descriptor. It should get this data descriptor from the io operation callback.

Life Cycle:

Passing in a data descriptor: A data descriptor is first created by the `globus` user. The user can add driver specific data descriptors to it. Once the user has created and set the attributes on its data descriptor to their liking they pass it into a `globusxio` data operation (either read or write). When the data descriptor is passed on `globus` makes an internal copy of it. It does this by first copying the user level data descriptor and then walking through the list of driver specific data descriptors contained in it and requesting the driver make a copy of the driver specific data descriptor. If ever a driver specific data descriptor is NULL `globus` need not call into its driver's `ddcopy` function. If ever the user level data descriptor is NULL `globus` need not deal with the data descriptor functionality at all.

A data descriptor coming back up the stack Once an io operation reaches the transport driver (the bottom of the stack) it takes on a slightly different role. On the way in it is describing what is requested to be done with the data, on the way out it is describing what has actually been done. Once the transport driver performs the operation it should adjust the data descriptor to reflect what has actually happened (few drivers will need to worry about this). Each driver on the way up can adjust the data descriptor and its driver specific data descriptor. When `xio` reaches the top of the stack it calls a user callback. When that callback returns all memory associated with the data descriptor is cleaned up. The interface function `globusxio_driver_data_descriptor_free()` is used for this.

## 9.2 Todo List

Global `globus.xio.http_acceptcallback(globus_xio_operation_t op, globus_result_t result, void *user_arg)`

When implemented in the XIO driver framework, parse the request header before returning from this, so the target is populated with meaningful information for the user. This will help enable persistent connections.



---

## Index

Attributes and Cntls [29](#), [36](#), [42](#), [47](#), [51](#), [63](#)

Driver Programming [17](#)

Driver Programming: String option [86](#)

driver\_pgm

- [globusxio\\_driver\\_attr\\_cntl.t](#), [19](#)
- [globusxio\\_driver\\_attr\\_copy.t](#), [18](#)
- [globusxio\\_driver\\_attr\\_destroy.t](#), [19](#)
- [globusxio\\_driver\\_attr\\_init.t](#), [18](#)
- [globusxio\\_driver\\_callback.t](#), [18](#)
- [globusxio\\_driver\\_closet](#), [21](#)
- [globusxio\\_driver\\_datacallback.t](#), [18](#)
- [globusxio\\_driver\\_eof\\_received](#), [24](#)
- [globusxio\\_driver\\_nished\\_accept](#), [22](#)
- [globusxio\\_driver\\_nished\\_close](#), [23](#)
- [globusxio\\_driver\\_nished\\_open](#), [22](#)
- [globusxio\\_driver\\_nished\\_read](#), [24](#)
- [globusxio\\_driver\\_nished\\_write](#), [25](#)
- [globusxio\\_driver\\_handlecntl](#), [22](#)
- [globusxio\\_driver\\_handlecntl.t](#), [21](#)
- [globusxio\\_driver\\_link\\_destroy.t](#), [20](#)
- [globusxio\\_driver\\_mergeoperation](#), [25](#)
- [globusxio\\_driver\\_operationcreate](#), [23](#)
- [globusxio\\_driver\\_operationis\\_blocking](#), [23](#)
- [globusxio\\_driver\\_passclose](#), [23](#)
- [globusxio\\_driver\\_passopen](#), [22](#)
- [globusxio\\_driver\\_passread](#), [24](#)
- [globusxio\\_driver\\_passwrite](#), [25](#)
- [globusxio\\_driver\\_read.t](#), [21](#)
- [globusxio\\_driver\\_serveraccept.t](#), [19](#)
- [globusxio\\_driver\\_servercntl.t](#), [20](#)
- [globusxio\\_driver\\_serverdestroy.t](#), [19](#)
- [globusxio\\_driver\\_serverinit.t](#), [19](#)
- [globusxio\\_driver\\_seteof\\_received](#), [24](#)
- [globusxio\\_driver\\_transformopent](#), [20](#)
- [globusxio\\_driver\\_transportopent](#), [20](#)
- [globusxio\\_driver\\_write.t](#), [21](#)

Env Variables [28](#), [42](#), [47](#), [51](#), [62](#)

Error Types [35](#), [41](#), [46](#), [49](#), [61](#), [69](#)

le\_driver\_cntls

- [GLOBUS\\_XIO\\_FILE\\_GET\\_BLOCKING\\_IO](#), [30](#)
- [GLOBUS\\_XIO\\_FILE\\_GET\\_FLAGS](#), [30](#)
- [GLOBUS\\_XIO\\_FILE\\_GET\\_HANDLE](#), [30](#)
- [GLOBUS\\_XIO\\_FILE\\_GET\\_MODE](#), [30](#)
- [GLOBUS\\_XIO\\_FILE\\_GET\\_TRUNC\\_OFFSET](#), [30](#)
- [GLOBUS\\_XIO\\_FILE\\_SEEK](#), [30](#)
- [GLOBUS\\_XIO\\_FILE\\_SET\\_BLOCKING\\_IO](#), [30](#)
- [GLOBUS\\_XIO\\_FILE\\_SET\\_FLAGS](#), [30](#)

[GLOBUS\\_XIO\\_FILE\\_SET\\_HANDLE](#), [30](#)

[GLOBUS\\_XIO\\_FILE\\_SET\\_MODE](#), [30](#)

[GLOBUS\\_XIO\\_FILE\\_SET\\_TRUNC\\_OFFSET](#), [30](#)

le\_driver\_cntls

- [globusxio\\_attr\\_cntl](#), [30–32](#)
- [globusxio\\_le\\_attr\\_cmd.t](#), [29](#)
- [globusxio\\_handlecntl](#), [31](#), [32](#)

le\_driver\_types

- [GLOBUS\\_XIO\\_FILE\\_APPEND](#), [34](#)
- [GLOBUS\\_XIO\\_FILE\\_BINARY](#), [34](#)
- [GLOBUS\\_XIO\\_FILE\\_CREAT](#), [34](#)
- [GLOBUS\\_XIO\\_FILE\\_EXCL](#), [34](#)
- [GLOBUS\\_XIO\\_FILE\\_IRGRP](#), [34](#)
- [GLOBUS\\_XIO\\_FILE\\_IROTH](#), [34](#)
- [GLOBUS\\_XIO\\_FILE\\_IRUSR](#), [34](#)
- [GLOBUS\\_XIO\\_FILE\\_IRWXG](#), [34](#)
- [GLOBUS\\_XIO\\_FILE\\_IRWXO](#), [34](#)
- [GLOBUS\\_XIO\\_FILE\\_IRWXU](#), [34](#)
- [GLOBUS\\_XIO\\_FILE\\_IWGRP](#), [34](#)
- [GLOBUS\\_XIO\\_FILE\\_IWOTH](#), [34](#)
- [GLOBUS\\_XIO\\_FILE\\_IWUSR](#), [34](#)
- [GLOBUS\\_XIO\\_FILE\\_IXGRP](#), [34](#)
- [GLOBUS\\_XIO\\_FILE\\_IXOTH](#), [34](#)
- [GLOBUS\\_XIO\\_FILE\\_IXUSR](#), [34](#)
- [GLOBUS\\_XIO\\_FILE\\_RDONLY](#), [34](#)
- [GLOBUS\\_XIO\\_FILE\\_RDWR](#), [34](#)
- [GLOBUS\\_XIO\\_FILE\\_SEEK\\_CUR](#), [34](#)
- [GLOBUS\\_XIO\\_FILE\\_SEEK\\_END](#), [34](#)
- [GLOBUS\\_XIO\\_FILE\\_SEEK\\_SET](#), [34](#)
- [GLOBUS\\_XIO\\_FILE\\_TEXT](#), [34](#)
- [GLOBUS\\_XIO\\_FILE\\_TRUNC](#), [34](#)
- [GLOBUS\\_XIO\\_FILE\\_WRONLY](#), [34](#)

le\_driver\_types

- [globusxio\\_le\\_ag.t](#), [33](#)
- [GLOBUS\\_XIO\\_FILE\\_INVALID\\_HANDLE](#), [33](#)
- [globusxio\\_le\\_modet](#), [34](#)
- [globusxio\\_le\\_whencet](#), [34](#)

Globus XIO Driver, [15](#)

Globus XIO File Driver, [27](#)

Globus XIO HTTP Driver, [35](#)

Globus XIO MODEE Driver, [41](#)

Globus XIO ORDERING Driver, [46](#)

Globus XIO TCP Driver, [50](#)

Globus XIO UDP Driver, [61](#)

globusi\_xio\_op\_type.e

- [GLOBUS\\_XIO\\_API](#), [7](#)

globusxio\_acceptcallback.t

- [GLOBUS\\_XIO\\_API](#), [6](#)

GLOBUS\_XIO\_API

---

- GLOBUS\_XIO\_GET\_LOCAL\_CONTACT, 7
  - GLOBUS\_XIO\_GET\_LOCAL\_NUMERIC\_CONTACT, 7
  - GLOBUS\_XIO\_GET\_REMOTE\_CONTACT, 7
  - GLOBUS\_XIO\_GET\_REMOTE\_NUMERIC\_CONTACT, 7
  - GLOBUS\_XIO\_SEEK, 7
  - GLOBUS\_XIO\_SET\_STRING\_OPTIONS, 7
- GLOBUS\_XIO\_API
  - globusi\_xio\_op\_type\_e, 7
  - globusxio\_acceptcallbackt, 6
  - globusxio\_attr\_cntl, 7
  - globusxio\_attr\_copy, 8
  - globusxio\_attr\_destroy, 8
  - globusxio\_attr\_init, 7
  - globusxio\_callbackt, 6
  - globusxio\_close, 12
  - globusxio\_datacallbackt, 6
  - globusxio\_datadescriptorcntl, 10
  - globusxio\_datadescriptordestroy, 10
  - globusxio\_datadescriptorinit, 9
  - globusxio\_handlecmdt, 7
  - globusxio\_handlecntl, 10, 13
  - globusxio\_handlecreate, 9
  - globusxio\_handlecreatefrom\_url, 12
  - globusxio\_iovec.callbackt, 6
  - globusxio\_open, 11
  - globusxio\_operationtype\_t, 7
  - globusxio\_read, 11
  - globusxio\_readv, 11
  - globusxio\_registerclose, 12
  - globusxio\_registeropen, 10
  - globusxio\_registerread, 11
  - globusxio\_registerreadv, 11
  - globusxio\_registerwrite, 11
  - globusxio\_registerwritev, 12
  - globusxio\_serveraccept, 9
  - globusxio\_servercallbackt, 6
  - globusxio\_serverclose, 9
  - globusxio\_servercntl, 9
  - globusxio\_servercreate, 8
  - globusxio\_serverget\_contactstring, 8
  - globusxio\_serverregisteraccept, 9
  - globusxio\_serverregisterclose, 9
  - globusxio\_stackcopy, 8
  - globusxio\_stackdestroy, 8
  - globusxio\_stackinit, 8
  - globusxio\_stackpushdriver, 8
  - globusxio\_timeoutcallbackt, 6
  - globusxio\_write, 12
  - globusxio\_writev, 12
- globusxio\_attr\_cntl
  - le\_driver\_cntls, 30–32
- GLOBUS\_XIO\_API, 7
  - globusxio\_http.h, 73
  - http\_driver\_cntls, 39, 40
  - modee\_driver\_cntls, 44–46
  - orderingdriver\_cntls, 48, 49
  - tcp\_driver\_cntls, 54–58
  - udp\_driver\_cntls, 65–67, 69
- globusxio\_attr\_copy
  - GLOBUS\_XIO\_API, 8
- globusxio\_attr\_destroy
  - GLOBUS\_XIO\_API, 8
- globusxio\_attr\_init
  - GLOBUS\_XIO\_API, 7
- globusxio\_callbackt
  - GLOBUS\_XIO\_API, 6
- globusxio\_close
  - GLOBUS\_XIO\_API, 12
- globusxio\_datacallbackt
  - GLOBUS\_XIO\_API, 6
- globusxio\_datadescriptorcntl
  - GLOBUS\_XIO\_API, 10
  - globusxio\_http.h, 73
  - modee\_driver\_cntls, 45
  - tcp\_driver\_cntls, 59
  - udp\_driver\_cntls, 68
- globusxio\_datadescriptordestroy
  - GLOBUS\_XIO\_API, 10
- globusxio\_datadescriptorinit
  - GLOBUS\_XIO\_API, 9
- globusxio\_driver\_attr\_cntl\_t
  - driver\_pgm, 19
- globusxio\_driver\_attr\_copy\_t
  - driver\_pgm, 18
- globusxio\_driver\_attr\_destroy\_t
  - driver\_pgm, 19
- globusxio\_driver\_attr\_init\_t
  - driver\_pgm, 18
- globusxio\_driver\_callbackt
  - driver\_pgm, 18
- globusxio\_driver\_close\_t
  - driver\_pgm, 21
- globusxio\_driver\_datacallbackt
  - driver\_pgm, 18
- globusxio\_driver\_eof\_received
  - driver\_pgm, 24
- globusxio\_driver\_nished\_accept
  - driver\_pgm, 22
- globusxio\_driver\_nished\_close
  - driver\_pgm, 23
- globusxio\_driver\_nished\_open
  - driver\_pgm, 22
- globusxio\_driver\_nished\_read
  - driver\_pgm, 24

- globusxio\_driver\_nished.write
  - driver\_pgm, [25](#)
- globusxio\_driver\_handlecntl
  - driver\_pgm, [22](#)
- globusxio\_driver\_handlecntl.t
  - driver\_pgm, [21](#)
- globusxio\_driver\_link\_destroyt
  - driver\_pgm, [20](#)
- globusxio\_driver\_mergeoperation
  - driver\_pgm, [25](#)
- globusxio\_driver\_operationcreate
  - driver\_pgm, [23](#)
- globusxio\_driver\_operationis\_blocking
  - driver\_pgm, [23](#)
- globusxio\_driver\_passclose
  - driver\_pgm, [23](#)
- globusxio\_driver\_passopen
  - driver\_pgm, [22](#)
- globusxio\_driver\_passread
  - driver\_pgm, [24](#)
- globusxio\_driver\_passwrite
  - driver\_pgm, [25](#)
- globusxio\_driver\_readt
  - driver\_pgm, [21](#)
- globusxio\_driver\_serveracceptt
  - driver\_pgm, [19](#)
- globusxio\_driver\_servercntl.t
  - driver\_pgm, [20](#)
- globusxio\_driver\_serverdestroyt
  - driver\_pgm, [19](#)
- globusxio\_driver\_serverinit.t
  - driver\_pgm, [19](#)
- globusxio\_driver\_seteof\_received
  - driver\_pgm, [24](#)
- globusxio\_driver\_transformopent
  - driver\_pgm, [20](#)
- globusxio\_driver\_transportopent
  - driver\_pgm, [20](#)
- globusxio\_driver\_writet
  - driver\_pgm, [21](#)
- GLOBUS\_XIO\_FILE\_APPEND
  - le\_driver.types, [34](#)
- globusxio\_le\_attr.cmd.t
  - le\_driver.cntls, [29](#)
- GLOBUS\_XIO\_FILE\_BINARY
  - le\_driver.types, [34](#)
- GLOBUS\_XIO\_FILE\_CREAT
  - le\_driver.types, [34](#)
- globusxio\_le\_driver.h, [71](#)
- GLOBUS\_XIO\_FILE\_EXCL
  - le\_driver.types, [34](#)
- globusxio\_le\_ag.t
  - le\_driver.types, [33](#)
- GLOBUS\_XIO\_FILE\_GET\_BLOCKING\_IO
  - le\_driver.cntls, [30](#)
- GLOBUS\_XIO\_FILE\_GET\_FLAGS
  - le\_driver.cntls, [30](#)
- GLOBUS\_XIO\_FILE\_GET\_HANDLE
  - le\_driver.cntls, [30](#)
- GLOBUS\_XIO\_FILE\_GET\_MODE
  - le\_driver.cntls, [30](#)
- GLOBUS\_XIO\_FILE\_GET\_TRUNC\_OFFSET
  - le\_driver.cntls, [30](#)
- GLOBUS\_XIO\_FILE\_INVALID\_HANDLE
  - le\_driver.types, [33](#)
- GLOBUS\_XIO\_FILE\_IRGRP
  - le\_driver.types, [34](#)
- GLOBUS\_XIO\_FILE\_IROTH
  - le\_driver.types, [34](#)
- GLOBUS\_XIO\_FILE\_IRUSR
  - le\_driver.types, [34](#)
- GLOBUS\_XIO\_FILE\_IRWXG
  - le\_driver.types, [34](#)
- GLOBUS\_XIO\_FILE\_IRWXO
  - le\_driver.types, [34](#)
- GLOBUS\_XIO\_FILE\_IRWXU
  - le\_driver.types, [34](#)
- GLOBUS\_XIO\_FILE\_IWGRP
  - le\_driver.types, [34](#)
- GLOBUS\_XIO\_FILE\_IWOTH
  - le\_driver.types, [34](#)
- GLOBUS\_XIO\_FILE\_IWUSR
  - le\_driver.types, [34](#)
- GLOBUS\_XIO\_FILE\_IXGRP
  - le\_driver.types, [34](#)
- GLOBUS\_XIO\_FILE\_IXOTH
  - le\_driver.types, [34](#)
- GLOBUS\_XIO\_FILE\_IXUSR
  - le\_driver.types, [34](#)
- globusxio\_le\_modet
  - le\_driver.types, [34](#)
- GLOBUS\_XIO\_FILE\_RDONLY
  - le\_driver.types, [34](#)
- GLOBUS\_XIO\_FILE\_RDWR
  - le\_driver.types, [34](#)
- GLOBUS\_XIO\_FILE\_SEEK
  - le\_driver.cntls, [30](#)
- GLOBUS\_XIO\_FILE\_SEEK\_CUR
  - le\_driver.types, [34](#)
- GLOBUS\_XIO\_FILE\_SEEK\_END
  - le\_driver.types, [34](#)
- GLOBUS\_XIO\_FILE\_SEEK\_SET
  - le\_driver.types, [34](#)
- GLOBUS\_XIO\_FILE\_SET\_BLOCKING\_IO
  - le\_driver.cntls, [30](#)
- GLOBUS\_XIO\_FILE\_SET\_FLAGS

- le\_driver\_cntls, [30](#)
- GLOBUS\_XIO\_FILE\_SET\_HANDLE
  - le\_driver\_cntls, [30](#)
- GLOBUS\_XIO\_FILE\_SET\_MODE
  - le\_driver\_cntls, [30](#)
- GLOBUS\_XIO\_FILE\_SET\_TRUNC\_OFFSET
  - le\_driver\_cntls, [30](#)
- GLOBUS\_XIO\_FILE\_TEXT
  - le\_driver\_types, [34](#)
- GLOBUS\_XIO\_FILE\_TRUNC
  - le\_driver\_types, [34](#)
- globusxio\_le\_whencet
  - le\_driver\_types, [34](#)
- GLOBUS\_XIO\_FILE\_WRONLY
  - le\_driver\_types, [34](#)
- GLOBUS\_XIO\_GET\_LOCAL\_CONTACT
  - GLOBUS\_XIO\_API, [7](#)
- GLOBUS\_XIO\_GET\_LOCAL\_NUMERIC\_CONTACT
  - GLOBUS\_XIO\_API, [7](#)
- GLOBUS\_XIO\_GET\_REMOTE\_CONTACT
  - GLOBUS\_XIO\_API, [7](#)
- GLOBUS\_XIO\_GET\_REMOTE\_NUMERIC\_CONTACT
  - GLOBUS\_XIO\_API, [7](#)
- globusxio\_handlecmdt
  - GLOBUS\_XIO\_API, [7](#)
- globusxio\_handlecntl
  - le\_driver\_cntls, [31](#), [32](#)
  - GLOBUS\_XIO\_API, [10](#), [13](#)
  - http\_driver\_cntls, [38](#), [39](#)
  - modee\_driver\_cntls, [45](#)
  - orderingdriver\_cntls, [48](#)
  - tcp\_driver\_cntls, [56–59](#)
  - udp\_driver\_cntls, [67](#), [68](#)
- globusxio\_handlecreate
  - GLOBUS\_XIO\_API, [9](#)
- globusxio\_handlecreatefrom\_url
  - GLOBUS\_XIO\_API, [12](#)
- globusxio\_http.h, [72](#)
  - globusxio\_attr\_cntl, [73](#)
  - globusxio\_datadescriptorcntl, [73](#)
- globusxio\_http\_attr\_cmdt
  - http\_driver\_cntls, [37](#)
- GLOBUS\_XIO\_HTTP\_ATTR\_DELAY\_WRITE\_HEADER
  - http\_driver\_cntls, [37](#)
- GLOBUS\_XIO\_HTTP\_ATTR\_SET\_REQUEST\_HEADER
  - http\_driver\_cntls, [37](#)
- GLOBUS\_XIO\_HTTP\_ATTR\_SET\_REQUEST\_HTTP\_VERSION
  - http\_driver\_cntls, [37](#)
- GLOBUS\_XIO\_HTTP\_ATTR\_SET\_REQUEST\_METHOD
  - http\_driver\_cntls, [37](#)
- GLOBUS\_XIO\_HTTP\_ERROR\_EOF
  - http\_driver\_errors, [41](#)
- GLOBUS\_XIO\_HTTP\_ERROR\_INVALID\_HEADER
  - http\_driver\_errors, [41](#)
- GLOBUS\_XIO\_HTTP\_ERROR\_NO\_ENTITY
  - http\_driver\_errors, [41](#)
- GLOBUS\_XIO\_HTTP\_ERROR\_PARSE
  - http\_driver\_errors, [41](#)
- GLOBUS\_XIO\_HTTP\_ERROR\_PERSISTENT\_CONNECTION\_DROPPED
  - http\_driver\_errors, [41](#)
- globusxio\_http\_errorst
  - http\_driver\_errors, [41](#)
- GLOBUS\_XIO\_HTTP\_GET\_REQUEST
  - http\_driver\_cntls, [37](#)
- GLOBUS\_XIO\_HTTP\_GET\_RESPONSE
  - http\_driver\_cntls, [37](#)
- globusxio\_http\_handlecmdt
  - http\_driver\_cntls, [37](#)
- GLOBUS\_XIO\_HTTP\_HANDLE\_SET\_END\_OF\_ENTITY
  - http\_driver\_cntls, [37](#)
- GLOBUS\_XIO\_HTTP\_HANDLE\_SET\_RESPONSE\_HEADER
  - http\_driver\_cntls, [37](#)
- GLOBUS\_XIO\_HTTP\_HANDLE\_SET\_RESPONSE\_HTTP\_VERSION
  - http\_driver\_cntls, [37](#)
- GLOBUS\_XIO\_HTTP\_HANDLE\_SET\_RESPONSE\_REASON\_PHRASE
  - http\_driver\_cntls, [37](#)
- GLOBUS\_XIO\_HTTP\_HANDLE\_SET\_RESPONSE\_STATUS\_CODE
  - http\_driver\_cntls, [37](#)
- globusxio\_http\_header, [70](#)
  - name, [70](#)
  - value, [70](#)
- GLOBUS\_XIO\_HTTP\_VERSION\_1\_0
  - http\_driver, [35](#)
- GLOBUS\_XIO\_HTTP\_VERSION\_1\_1
  - http\_driver, [35](#)
- globusxio\_http\_versiont
  - http\_driver, [35](#)
- globusxio\_iovec\_callbackt
  - GLOBUS\_XIO\_API, [6](#)
- globusxio\_modee\_cmdt
  - modee\_driver\_cntls, [43](#)
- GLOBUS\_XIO\_MODE\_E\_DD\_GET\_OFFSET
  - modee\_driver\_cntls, [44](#)
- globusxio\_modee\_driver.h, [74](#)

- globusxio\_mode\_e\_error\_type\_t
  - mode\_e\_driver\_errors, [46](#)
- GLOBUS\_XIO\_MODE\_E\_GET\_MANUAL\_EODC
  - mode\_e\_driver\_cntls, [43](#)
- GLOBUS\_XIO\_MODE\_E\_GET\_NUM\_STREAMS
  - mode\_e\_driver\_cntls, [43](#)
- GLOBUS\_XIO\_MODE\_E\_GET\_OFFSETREADS
  - mode\_e\_driver\_cntls, [43](#)
- GLOBUS\_XIO\_MODE\_E\_GET\_STACK
  - mode\_e\_driver\_cntls, [43](#)
- GLOBUS\_XIO\_MODE\_E\_GET\_STACK\_ATTR
  - mode\_e\_driver\_cntls, [44](#)
- GLOBUS\_XIO\_MODE\_E\_HEADER\_ERROR
  - mode\_e\_driver\_errors, [46](#)
- GLOBUS\_XIO\_MODE\_E\_SENDEOD
  - mode\_e\_driver\_cntls, [43](#)
- GLOBUS\_XIO\_MODE\_E\_SETEODC
  - mode\_e\_driver\_cntls, [44](#)
- GLOBUS\_XIO\_MODE\_E\_SET\_MANUAL\_EODC
  - mode\_e\_driver\_cntls, [43](#)
- GLOBUS\_XIO\_MODE\_E\_SET\_NUM\_STREAMS
  - mode\_e\_driver\_cntls, [43](#)
- GLOBUS\_XIO\_MODE\_E\_SET\_OFFSETREADS
  - mode\_e\_driver\_cntls, [43](#)
- GLOBUS\_XIO\_MODE\_E\_SET\_STACK
  - mode\_e\_driver\_cntls, [43](#)
- GLOBUS\_XIO\_MODE\_E\_SET\_STACK\_ATTR
  - mode\_e\_driver\_cntls, [44](#)
- globusxio\_open
  - GLOBUS\_XIO\_API, [11](#)
- globusxio\_operation\_type\_t
  - GLOBUS\_XIO\_API, [7](#)
- globusxio\_ordering\_cmd\_t
  - ordering\_driver\_cntls, [48](#)
- globusxio\_ordering\_driver.h, [75](#)
- GLOBUS\_XIO\_ORDERING\_ERROR\_CANCEL
  - ordering\_driver\_errors, [50](#)
- GLOBUS\_XIO\_ORDERING\_ERROR\_READ
  - ordering\_driver\_errors, [50](#)
- globusxio\_ordering\_error\_type\_t
  - ordering\_driver\_errors, [50](#)
- GLOBUS\_XIO\_ORDERING\_GET\_BUF\_SIZE
  - ordering\_driver\_cntls, [48](#)
- GLOBUS\_XIO\_ORDERING\_GET\_BUFFERING
  - ordering\_driver\_cntls, [48](#)
- GLOBUS\_XIO\_ORDERING\_GET\_MAX\_BUF\_COUNT
  - ordering\_driver\_cntls, [48](#)
- GLOBUS\_XIO\_ORDERING\_GET\_MAX\_READ\_COUNT
  - ordering\_driver\_cntls, [48](#)
- GLOBUS\_XIO\_ORDERING\_SET\_BUF\_SIZE
  - ordering\_driver\_cntls, [48](#)
- GLOBUS\_XIO\_ORDERING\_SET\_BUFFERING
  - ordering\_driver\_cntls, [48](#)
- GLOBUS\_XIO\_ORDERING\_SET\_MAX\_BUF\_COUNT
  - ordering\_driver\_cntls, [48](#)
- GLOBUS\_XIO\_ORDERING\_SET\_MAX\_READ\_COUNT
  - ordering\_driver\_cntls, [48](#)
- GLOBUS\_XIO\_ORDERING\_SET\_OFFSET
  - ordering\_driver\_cntls, [48](#)
- globusxio\_read
  - GLOBUS\_XIO\_API, [11](#)
- globusxio\_readv
  - GLOBUS\_XIO\_API, [11](#)
- globusxio\_register\_close
  - GLOBUS\_XIO\_API, [12](#)
- globusxio\_register\_open
  - GLOBUS\_XIO\_API, [10](#)
- globusxio\_register\_read
  - GLOBUS\_XIO\_API, [11](#)
- globusxio\_register\_readv
  - GLOBUS\_XIO\_API, [11](#)
- globusxio\_register\_write
  - GLOBUS\_XIO\_API, [11](#)
- globusxio\_register\_writev
  - GLOBUS\_XIO\_API, [12](#)
- GLOBUS\_XIO\_SEEK
  - GLOBUS\_XIO\_API, [7](#)
- globusxio\_server\_accept
  - GLOBUS\_XIO\_API, [9](#)
- globusxio\_server\_callback\_t
  - GLOBUS\_XIO\_API, [6](#)
- globusxio\_server\_close
  - GLOBUS\_XIO\_API, [9](#)
- globusxio\_server\_cntl
  - GLOBUS\_XIO\_API, [9](#)
  - tcp\_driver\_cntls, [56](#), [60](#)
- globusxio\_server\_create
  - GLOBUS\_XIO\_API, [8](#)
- globusxio\_server\_get\_contact\_string
  - GLOBUS\_XIO\_API, [8](#)
- globusxio\_server\_register\_accept
  - GLOBUS\_XIO\_API, [9](#)
- globusxio\_server\_register\_close
  - GLOBUS\_XIO\_API, [9](#)
- GLOBUS\_XIO\_SET\_STRING\_OPTIONS
  - GLOBUS\_XIO\_API, [7](#)
- globusxio\_stack\_copy
  - GLOBUS\_XIO\_API, [8](#)
- globusxio\_stack\_destroy
  - GLOBUS\_XIO\_API, [8](#)
- globusxio\_stack\_init
  - GLOBUS\_XIO\_API, [8](#)

globusxio\_stackpushdriver  
     GLOBUS\_XIO\_API, [8](#)  
 globusxio\_string.cntl.bool  
     string.driver.pgm, [27](#)  
 globusxio\_string.cntl.bouncer  
     string.driver.pgm, [27](#)  
 globusxio\_string.cntl.oat  
     string.driver.pgm, [27](#)  
 globusxio\_string.cntl.int  
     string.driver.pgm, [27](#)  
 globusxio\_string.cntl.int.int  
     string.driver.pgm, [27](#)  
 globusxio\_string.cntl.string  
     string.driver.pgm, [27](#)  
 GLOBUS\_XIO\_TCP\_AFFECT\_ATTR\_DEFAULTS  
     tcp.driver.cntls, [54](#)  
 globusxio\_tcp.cmd.t  
     tcp.driver.cntls, [53](#)  
 globusxio\_tcp.driver.h, [75](#)  
 GLOBUS\_XIO\_TCP\_ERRORNO\_ADDRS  
     tcp.driver.errors, [61](#)  
 globusxio\_tcp.error.type.t  
     tcp.driver.errors, [61](#)  
 GLOBUS\_XIO\_TCP\_GET\_BACKLOG  
     tcp.driver.cntls, [53](#)  
 GLOBUS\_XIO\_TCP\_GET\_BLOCKING\_IO  
     tcp.driver.cntls, [54](#)  
 GLOBUS\_XIO\_TCP\_GET\_CONNECT\_RANGE  
     tcp.driver.cntls, [53](#)  
 GLOBUS\_XIO\_TCP\_GET\_HANDLE  
     tcp.driver.cntls, [53](#)  
 GLOBUS\_XIO\_TCP\_GET\_INTERFACE  
     tcp.driver.cntls, [53](#)  
 GLOBUS\_XIO\_TCP\_GET\_KEEPA\_LIVE  
     tcp.driver.cntls, [53](#)  
 GLOBUS\_XIO\_TCP\_GET\_LINGER  
     tcp.driver.cntls, [54](#)  
 GLOBUS\_XIO\_TCP\_GET\_LISTEN\_RANGE  
     tcp.driver.cntls, [53](#)  
 GLOBUS\_XIO\_TCP\_GET\_LOCAL\_CONTACT  
     tcp.driver.cntls, [54](#)  
 GLOBUS\_XIO\_TCP\_GET\_LOCAL\_NUMERIC\_CONTACT  
     tcp.driver.cntls, [54](#)  
 GLOBUS\_XIO\_TCP\_GET\_NO\_IPV6  
     tcp.driver.cntls, [53](#)  
 GLOBUS\_XIO\_TCP\_GET\_NODELAY  
     tcp.driver.cntls, [54](#)  
 GLOBUS\_XIO\_TCP\_GET\_OOBLINE  
     tcp.driver.cntls, [54](#)  
 GLOBUS\_XIO\_TCP\_GET\_PORT  
     tcp.driver.cntls, [53](#)  
 GLOBUS\_XIO\_TCP\_GET\_RCVBUF  
     tcp.driver.cntls, [54](#)  
 GLOBUS\_XIO\_TCP\_GET\_REMOTE\_CONTACT  
     tcp.driver.cntls, [54](#)  
 GLOBUS\_XIO\_TCP\_GET\_REMOTE\_NUMERIC\_CONTACT  
     tcp.driver.cntls, [54](#)  
 GLOBUS\_XIO\_TCP\_GET\_RESTRICTPORT  
     tcp.driver.cntls, [53](#)  
 GLOBUS\_XIO\_TCP\_GET\_REUSEADDR  
     tcp.driver.cntls, [53](#)  
 GLOBUS\_XIO\_TCP\_GET\_SEND\_FLAGS  
     tcp.driver.cntls, [54](#)  
 GLOBUS\_XIO\_TCP\_GET\_SERVICE  
     tcp.driver.cntls, [53](#)  
 GLOBUS\_XIO\_TCP\_GET\_SNDBUF  
     tcp.driver.cntls, [54](#)  
 GLOBUS\_XIO\_TCP\_INVALID\_HANDLE  
     tcp.driver.types, [60](#)  
 globusxio\_tcp.send.ags.t  
     tcp.driver.types, [61](#)  
 GLOBUS\_XIO\_TCP\_SEND\_OOB  
     tcp.driver.types, [61](#)  
 GLOBUS\_XIO\_TCP\_SET\_BACKLOG  
     tcp.driver.cntls, [53](#)  
 GLOBUS\_XIO\_TCP\_SET\_BLOCKING\_IO  
     tcp.driver.cntls, [54](#)  
 GLOBUS\_XIO\_TCP\_SET\_CONNECT\_RANGE  
     tcp.driver.cntls, [53](#)  
 GLOBUS\_XIO\_TCP\_SET\_HANDLE  
     tcp.driver.cntls, [53](#)  
 GLOBUS\_XIO\_TCP\_SET\_INTERFACE  
     tcp.driver.cntls, [53](#)  
 GLOBUS\_XIO\_TCP\_SET\_KEEPA\_LIVE  
     tcp.driver.cntls, [53](#)  
 GLOBUS\_XIO\_TCP\_SET\_LINGER  
     tcp.driver.cntls, [54](#)  
 GLOBUS\_XIO\_TCP\_SET\_LISTEN\_RANGE  
     tcp.driver.cntls, [53](#)  
 GLOBUS\_XIO\_TCP\_SET\_NO\_IPV6  
     tcp.driver.cntls, [53](#)  
 GLOBUS\_XIO\_TCP\_SET\_NODELAY  
     tcp.driver.cntls, [54](#)  
 GLOBUS\_XIO\_TCP\_SET\_OOBLINE  
     tcp.driver.cntls, [54](#)  
 GLOBUS\_XIO\_TCP\_SET\_PORT  
     tcp.driver.cntls, [53](#)  
 GLOBUS\_XIO\_TCP\_SET\_RCVBUF  
     tcp.driver.cntls, [54](#)  
 GLOBUS\_XIO\_TCP\_SET\_RESTRICTPORT  
     tcp.driver.cntls, [53](#)  
 GLOBUS\_XIO\_TCP\_SET\_REUSEADDR  
     tcp.driver.cntls, [53](#)  
 GLOBUS\_XIO\_TCP\_SET\_SEND\_FLAGS



- tcp.driver.cntls, [54](#)
- GLOBUS\_XIO\_TCP\_SET\_SERVICE
  - tcp.driver.cntls, [53](#)
- GLOBUS\_XIO\_TCP\_SET\_SNDBUF
  - tcp.driver.cntls, [54](#)
- globusxio\_timeout\_callback\_t
  - GLOBUS\_XIO\_API, [6](#)
- globusxio\_udp\_cmd\_t
  - udp.driver.cntls, [64](#)
- GLOBUS\_XIO\_UDP\_CONNECT
  - udp.driver.cntls, [64](#)
- globusxio\_udp\_driver.h, [77](#)
- GLOBUS\_XIO\_UDP\_ERROR\_NO\_ADDRS
  - udp.driver.errors, [70](#)
- GLOBUS\_XIO\_UDP\_ERROR\_SHORT\_WRITE
  - udp.driver.errors, [70](#)
- globusxio\_udp\_error\_type\_t
  - udp.driver.errors, [70](#)
- GLOBUS\_XIO\_UDP\_GET\_CONTACT
  - udp.driver.cntls, [64](#)
- GLOBUS\_XIO\_UDP\_GET\_HANDLE
  - udp.driver.cntls, [64](#)
- GLOBUS\_XIO\_UDP\_GET\_INTERFACE
  - udp.driver.cntls, [64](#)
- GLOBUS\_XIO\_UDP\_GET\_LISTEN\_RANGE
  - udp.driver.cntls, [64](#)
- GLOBUS\_XIO\_UDP\_GET\_NO\_IPV6
  - udp.driver.cntls, [64](#)
- GLOBUS\_XIO\_UDP\_GET\_NUMERIC\_CONTACT
  - udp.driver.cntls, [64](#)
- GLOBUS\_XIO\_UDP\_GET\_PORT
  - udp.driver.cntls, [64](#)
- GLOBUS\_XIO\_UDP\_GET\_RCVBUF
  - udp.driver.cntls, [64](#)
- GLOBUS\_XIO\_UDP\_GET\_RESTRICTPORT
  - udp.driver.cntls, [64](#)
- GLOBUS\_XIO\_UDP\_GET\_REUSEADDR
  - udp.driver.cntls, [64](#)
- GLOBUS\_XIO\_UDP\_GET\_SERVICE
  - udp.driver.cntls, [64](#)
- GLOBUS\_XIO\_UDP\_GET\_SNDBUF
  - udp.driver.cntls, [64](#)
- GLOBUS\_XIO\_UDP\_INVALID\_HANDLE
  - udp.driver.types, [69](#)
- GLOBUS\_XIO\_UDP\_SET\_CONTACT
  - udp.driver.cntls, [64](#)
- GLOBUS\_XIO\_UDP\_SET\_HANDLE
  - udp.driver.cntls, [64](#)
- GLOBUS\_XIO\_UDP\_SET\_INTERFACE
  - udp.driver.cntls, [64](#)
- GLOBUS\_XIO\_UDP\_SET\_LISTEN\_RANGE
  - udp.driver.cntls, [64](#)
- GLOBUS\_XIO\_UDP\_SET\_MULTICAST
  - udp.driver.cntls, [64](#)
- GLOBUS\_XIO\_UDP\_SET\_NO\_IPV6
  - udp.driver.cntls, [64](#)
- GLOBUS\_XIO\_UDP\_SET\_PORT
  - udp.driver.cntls, [64](#)
- GLOBUS\_XIO\_UDP\_SET\_RCVBUF
  - udp.driver.cntls, [64](#)
- GLOBUS\_XIO\_UDP\_SET\_RESTRICTPORT
  - udp.driver.cntls, [64](#)
- GLOBUS\_XIO\_UDP\_SET\_REUSEADDR
  - udp.driver.cntls, [64](#)
- GLOBUS\_XIO\_UDP\_SET\_SERVICE
  - udp.driver.cntls, [64](#)
- GLOBUS\_XIO\_UDP\_SET\_SNDBUF
  - udp.driver.cntls, [64](#)
- globusxio\_write
  - GLOBUS\_XIO\_API, [12](#)
- globusxio\_writew
  - GLOBUS\_XIO\_API, [12](#)
- http\_driver
  - GLOBUS\_XIO\_HTTP\_VERSION\_1\_0, [35](#)
  - GLOBUS\_XIO\_HTTP\_VERSION\_1\_1, [35](#)
- http\_driver
  - globusxio\_http\_version\_t, [35](#)
- http\_driver.cntls
  - GLOBUS\_XIO\_HTTP\_ATTR\_DELAY\_WRITE\_HEADER, [37](#)
  - GLOBUS\_XIO\_HTTP\_ATTR\_SET\_REQUEST\_HEADER, [37](#)
  - GLOBUS\_XIO\_HTTP\_ATTR\_SET\_REQUEST\_HTTP\_VERSION, [37](#)
  - GLOBUS\_XIO\_HTTP\_ATTR\_SET\_REQUEST\_METHOD, [37](#)
  - GLOBUS\_XIO\_HTTP\_GET\_REQUEST, [37](#)
  - GLOBUS\_XIO\_HTTP\_GET\_RESPONSE, [37](#)
  - GLOBUS\_XIO\_HTTP\_HANDLE\_SET\_END\_OF\_ENTITY, [37](#)
  - GLOBUS\_XIO\_HTTP\_HANDLE\_SET\_RESPONSE\_HEADER, [37](#)
  - GLOBUS\_XIO\_HTTP\_HANDLE\_SET\_RESPONSE\_HTTP\_VERSION, [37](#)
  - GLOBUS\_XIO\_HTTP\_HANDLE\_SET\_RESPONSE\_REASON\_PHRASE, [37](#)
  - GLOBUS\_XIO\_HTTP\_HANDLE\_SET\_RESPONSE\_STATUS\_CODE, [37](#)
- http\_driver.cntls
  - globusxio\_attr\_cntl, [39](#), [40](#)
  - globusxio\_handle\_cntl, [38](#), [39](#)
  - globusxio\_http\_attr\_cmd\_t, [37](#)
  - globusxio\_http\_handlecmd\_t, [37](#)
- http\_driver.errors
  - GLOBUS\_XIO\_HTTP\_ERROR\_EOF, [41](#)

- GLOBUS\_XIO\_HTTP\_ERROR\_INVALID \_-  
HEADER, [41](#)
  - GLOBUS\_XIO\_HTTP\_ERROR\_NO\_ENTITY,  
[41](#)
  - GLOBUS\_XIO\_HTTP\_ERROR\_PARSE, [41](#)
  - GLOBUS\_XIO\_HTTP\_ERROR\_PERSISTENT-  
CONNECTION\_DROPPED, [41](#)
- http\_driver\_errors
  - globusxio\_http\_errorst, [41](#)
- mode\_e\_driver\_cntls
  - GLOBUS\_XIO\_MODE\_E\_DD\_GET\_OFFSET,  
[44](#)
  - GLOBUS\_XIO\_MODE\_E\_GET\_MANUAL \_-  
EODC, [43](#)
  - GLOBUS\_XIO\_MODE\_E\_GET\_NUM \_-  
STREAMS, [43](#)
  - GLOBUS\_XIO\_MODE\_E\_GET\_OFFSET-  
READS, [43](#)
  - GLOBUS\_XIO\_MODE\_E\_GET\_STACK, [43](#)
  - GLOBUS\_XIO\_MODE\_E\_GET\_STACK\_ATTR,  
[44](#)
  - GLOBUS\_XIO\_MODE\_E\_SEND\_EOD, [43](#)
  - GLOBUS\_XIO\_MODE\_E\_SET\_EODC, [44](#)
  - GLOBUS\_XIO\_MODE\_E\_SET\_MANUAL \_-  
EODC, [43](#)
  - GLOBUS\_XIO\_MODE\_E\_SET\_NUM \_-  
STREAMS, [43](#)
  - GLOBUS\_XIO\_MODE\_E\_SET\_OFFSET-  
READS, [43](#)
  - GLOBUS\_XIO\_MODE\_E\_SET\_STACK, [43](#)
  - GLOBUS\_XIO\_MODE\_E\_SET\_STACK\_ATTR,  
[44](#)
- mode\_e\_driver\_cntls
  - globusxio\_attr\_cntl, [44–46](#)
  - globusxio\_data\_descriptor\_cntl, [45](#)
  - globusxio\_handle\_cntl, [45](#)
  - globusxio\_mode\_e\_cmdt, [43](#)
- mode\_e\_driver\_errors
  - GLOBUS\_XIO\_MODE\_E\_HEADER\_ERROR,  
[46](#)
- mode\_e\_driver\_errors
  - globusxio\_mode\_e\_error\_type\_t, [46](#)
- name
  - globusxio\_http\_header\_t, [70](#)
- Opening/Closing, [28, 36, 42, 47, 50, 62](#)
- ordering\_driver\_cntls
  - GLOBUS\_XIO\_ORDERING\_GET\_BUF\_SIZE,  
[48](#)
  - GLOBUS\_XIO\_ORDERING\_GET \_-  
BUFFERING, [48](#)
- GLOBUS\_XIO\_ORDERING\_GET\_MAX\_BUF \_-  
COUNT, [48](#)
- GLOBUS\_XIO\_ORDERING\_GET\_MAX \_-  
READ\_COUNT, [48](#)
- GLOBUS\_XIO\_ORDERING\_SET\_BUF\_SIZE,  
[48](#)
- GLOBUS\_XIO\_ORDERING\_SET \_-  
BUFFERING, [48](#)
- GLOBUS\_XIO\_ORDERING\_SET\_MAX\_BUF \_-  
COUNT, [48](#)
- GLOBUS\_XIO\_ORDERING\_SET\_MAX \_-  
READ\_COUNT, [48](#)
- GLOBUS\_XIO\_ORDERING\_SET\_OFFSET, [48](#)
- ordering\_driver\_cntls
  - globusxio\_attr\_cntl, [48, 49](#)
  - globusxio\_handle\_cntl, [48](#)
  - globusxio\_ordering\_cmdt, [48](#)
- ordering\_driver\_errors
  - GLOBUS\_XIO\_ORDERING\_ERROR-  
CANCEL, [50](#)
  - GLOBUS\_XIO\_ORDERING\_ERROR\_READ,  
[50](#)
- ordering\_driver\_errors
  - globusxio\_ordering\_error\_type\_t, [50](#)
- Reading/Writing, [28, 36, 42, 47, 50, 62](#)
- Server, [36, 42, 51](#)
- string\_driver\_pgm
  - globusxio\_string\_cntl\_bool, [27](#)
  - globusxio\_string\_cntl\_bouncer, [27](#)
  - globusxio\_string\_cntl\_oat, [27](#)
  - globusxio\_string\_cntl\_int, [27](#)
  - globusxio\_string\_cntl\_int\_int, [27](#)
  - globusxio\_string\_cntl\_string, [27](#)
- tcp\_driver\_cntls
  - GLOBUS\_XIO\_TCP\_AFFECT\_ATTR \_-  
DEFAULTS, [54](#)
  - GLOBUS\_XIO\_TCP\_GET\_BACKLOG, [53](#)
  - GLOBUS\_XIO\_TCP\_GET\_BLOCKING\_IO, [54](#)
  - GLOBUS\_XIO\_TCP\_GET\_CONNECT-  
RANGE, [53](#)
  - GLOBUS\_XIO\_TCP\_GET\_HANDLE, [53](#)
  - GLOBUS\_XIO\_TCP\_GET\_INTERFACE, [53](#)
  - GLOBUS\_XIO\_TCP\_GET\_KEEPALIVE, [53](#)
  - GLOBUS\_XIO\_TCP\_GET\_LINGER, [54](#)
  - GLOBUS\_XIO\_TCP\_GET\_LISTEN\_RANGE, [53](#)
  - GLOBUS\_XIO\_TCP\_GET\_LOCAL\_CONTACT,  
[54](#)
  - GLOBUS\_XIO\_TCP\_GET\_LOCAL \_-  
NUMERIC\_CONTACT, [54](#)
  - GLOBUS\_XIO\_TCP\_GET\_NO\_IPV6, [53](#)
  - GLOBUS\_XIO\_TCP\_GET\_NODELAY, [54](#)



- GLOBUS\_XIO\_TCP\_GET\_OOBLINE, 54
- GLOBUS\_XIO\_TCP\_GET\_PORT, 53
- GLOBUS\_XIO\_TCP\_GET\_RCVBUF, 54
- GLOBUS\_XIO\_TCP\_GET\_REMOTE-  
CONTACT, 54
- GLOBUS\_XIO\_TCP\_GET\_REMOTE-  
NUMERIC\_CONTACT, 54
- GLOBUS\_XIO\_TCP\_GET\_RESTRICTPORT,  
53
- GLOBUS\_XIO\_TCP\_GET\_REUSEADDR, 53
- GLOBUS\_XIO\_TCP\_GET\_SEND\_FLAGS, 54
- GLOBUS\_XIO\_TCP\_GET\_SERVICE, 53
- GLOBUS\_XIO\_TCP\_GET\_SNDBUF, 54
- GLOBUS\_XIO\_TCP\_SET\_BACKLOG, 53
- GLOBUS\_XIO\_TCP\_SET\_BLOCKING\_IO, 54
- GLOBUS\_XIO\_TCP\_SET\_CONNECT\_RANGE,  
53
- GLOBUS\_XIO\_TCP\_SET\_HANDLE, 53
- GLOBUS\_XIO\_TCP\_SET\_INTERFACE, 53
- GLOBUS\_XIO\_TCP\_SET\_KEEPAIVE, 53
- GLOBUS\_XIO\_TCP\_SET\_LINGER, 54
- GLOBUS\_XIO\_TCP\_SET\_LISTEN\_RANGE, 53
- GLOBUS\_XIO\_TCP\_SET\_NO\_IPV6, 53
- GLOBUS\_XIO\_TCP\_SET\_NODELAY, 54
- GLOBUS\_XIO\_TCP\_SET\_OOBLINE, 54
- GLOBUS\_XIO\_TCP\_SET\_PORT, 53
- GLOBUS\_XIO\_TCP\_SET\_RCVBUF, 54
- GLOBUS\_XIO\_TCP\_SET\_RESTRICTPORT,  
53
- GLOBUS\_XIO\_TCP\_SET\_REUSEADDR, 53
- GLOBUS\_XIO\_TCP\_SET\_SEND\_FLAGS, 54
- GLOBUS\_XIO\_TCP\_SET\_SERVICE, 53
- GLOBUS\_XIO\_TCP\_SET\_SNDBUF, 54
- tcp\_driver\_cntls
  - globusxio\_attr\_cntl, 54–58
  - globusxio\_datadescriptorcntl, 59
  - globusxio\_handlecntl, 56–59
  - globusxio\_servercntl, 56, 60
  - globusxio\_tcp\_cmd.t, 53
- tcp\_driver\_errors
  - GLOBUS\_XIO\_TCP\_ERRORNO\_ADDRS, 61
- tcp\_driver\_errors
  - globusxio\_tcp\_error.type.t, 61
- tcp\_driver\_types
  - GLOBUS\_XIO\_TCP\_SEND\_OOB, 61
- tcp\_driver\_types
  - GLOBUS\_XIO\_TCP\_INVALID\_HANDLE, 60
  - globusxio\_tcp\_send\_ag.s.t, 61
- The globusxio user API, 4
- Types, 33, 46, 49, 60, 69
- udp\_driver\_cntls
  - GLOBUS\_XIO\_UDP\_CONNECT, 64
  - GLOBUS\_XIO\_UDP\_GET\_CONTACT, 64
  - GLOBUS\_XIO\_UDP\_GET\_HANDLE, 64
  - GLOBUS\_XIO\_UDP\_GET\_INTERFACE, 64
  - GLOBUS\_XIO\_UDP\_GET\_LISTEN\_RANGE,  
64
  - GLOBUS\_XIO\_UDP\_GET\_NO\_IPV6, 64
  - GLOBUS\_XIO\_UDP\_GET\_NUMERIC-  
CONTACT, 64
  - GLOBUS\_XIO\_UDP\_GET\_PORT, 64
  - GLOBUS\_XIO\_UDP\_GET\_RCVBUF, 64
  - GLOBUS\_XIO\_UDP\_GET\_RESTRICTPORT,  
64
  - GLOBUS\_XIO\_UDP\_GET\_REUSEADDR, 64
  - GLOBUS\_XIO\_UDP\_GET\_SERVICE, 64
  - GLOBUS\_XIO\_UDP\_GET\_SNDBUF, 64
  - GLOBUS\_XIO\_UDP\_GET\_CONTACT, 64
  - GLOBUS\_XIO\_UDP\_GET\_HANDLE, 64
  - GLOBUS\_XIO\_UDP\_GET\_INTERFACE, 64
  - GLOBUS\_XIO\_UDP\_GET\_LISTEN\_RANGE, 64
  - GLOBUS\_XIO\_UDP\_GET\_MULTICAST, 64
  - GLOBUS\_XIO\_UDP\_GET\_NO\_IPV6, 64
  - GLOBUS\_XIO\_UDP\_GET\_PORT, 64
  - GLOBUS\_XIO\_UDP\_GET\_RCVBUF, 64
  - GLOBUS\_XIO\_UDP\_GET\_RESTRICTPORT,  
64
  - GLOBUS\_XIO\_UDP\_GET\_REUSEADDR, 64
  - GLOBUS\_XIO\_UDP\_GET\_SERVICE, 64
  - GLOBUS\_XIO\_UDP\_GET\_SNDBUF, 64
- udp\_driver\_cntls
  - globusxio\_attr\_cntl, 65–67, 69
  - globusxio\_datadescriptorcntl, 68
  - globusxio\_handlecntl, 67, 68
  - globusxio\_udp\_cmd.t, 64
- udp\_driver\_errors
  - GLOBUS\_XIO\_UDP\_ERRORNO\_ADDRS, 70
  - GLOBUS\_XIO\_UDP\_ERROR\_SHORT\_WRITE,  
70
- udp\_driver\_errors
  - globusxio\_udp\_error.type.t, 70
- udp\_driver\_types
  - GLOBUS\_XIO\_UDP\_INVALID\_HANDLE, 69
- User API Assistance, 13
- value
  - globusxio\_http\_header.t, 70