

GT 4.0 XIO

GT 4.0 XIO

Table of Contents

1. Key Concepts	1
1. Overview	1
2. Conceptual Details	1
3. Related Documents	6
2. System Administrator's Guide	7
1. Introduction	7
2. Building and Installing	7
3. Configuring	7
4. Deploying	7
5. Testing	7
6. Security Considerations	7
7. Troubleshooting	8
3. User's Guide	9
1. Introduction	9
2. Command-line tools	9
3. Graphical user interfaces	9
4. Troubleshooting	9
4. Developer's Guide	10
1. Introduction	10
2. Before you begin	10
3. Architecture and design overview	12
4. Public interface	15
5. Usage scenarios	15
6. Tutorials	23
7. Debugging	23
8. Troubleshooting	24
9. Related Documentation	24
5. Fact Sheet	25
1. Brief component overview	25
2. Summary of features	25
3. Usability summary	26
4. Backward compatibility summary	26
5. Technology dependencies	26
6. Tested platforms	26
7. Associated standards	27
8. For more information	27
6. Public Interface Guide	28
1. Semantics and Syntax of XIO APIs	28
2. Semantics and Syntax of the WSDL	28
3. Command-Line Tools	28
4. Overview of Graphical User Interface	28
5. Semantics and Syntax of Domain-Specific Interface	28
6. Configuration Interface	28
7. Environment Variable Interface	29
7. Quality Profile	30
1. Test coverage reports	30
2. Code analysis reports	30
3. Outstanding bugs	30
4. Bug Fixes	30
5. Performance reports	30
8. Migrating Guide	31

1. Migrating from GT2	31
2. Migrating from GT3	31
9. 4.0.8 Release Notes	32
1. Introduction	32
2. Changes Summary	32
3. Bug Fixes	32
4. Known Problems	32
5. For More Information	32
10. 4.0.7 Release Notes	33
1. Introduction	33
2. Changes Summary	33
3. Bug Fixes	33
4. Known Problems	33
5. For More Information	33
11. 4.0.6 Release Notes	34
1. Introduction	34
2. Changes Summary	34
3. Bug Fixes	34
4. Known Problems	34
5. For More Information	34
12. 4.0.5 Release Notes	35
1. Introduction	35
2. Changes Summary	35
3. Bug Fixes	35
4. Known Problems	35
5. For More Information	36
13. 4.0.4 Release Notes	37
1. Introduction	37
2. Changes Summary	37
3. Bug Fixes	37
4. Known Problems	37
5. For More Information	38
14. 4.0.3 Release Notes	39
1. Introduction	39
2. Changes Summary	39
3. Bug Fixes	39
4. Known Problems	39
5. For More Information	39
15. 4.0.2 Release Notes	40
1. Introduction	40
2. Changes Summary	40
3. Bug Fixes	40
4. Known Problems	40
5. For More Information	40
16. 4.0.1 Release Notes	41
1. Introduction	41
2. Changes Summary	41
3. Bug Fixes	41
4. Known Problems	41
5. For More Information	41
17. 4.0.0 Release Notes	42
1. Component Overview	42
2. Feature Summary	42
3. Bug Fixes	43

4. Known Problems	43
5. Technology Dependencies	43
6. Tested Platforms	43
7. Backward Compatibility Summary	44
8. For More Information	44

Chapter 1. GT 4.0 Common Runtime Components: Key Concepts

1. Overview

The common runtime components provide GT4 web and pre-web services with a set of libraries and tools that allows these services to be platform independent, to build on various abstraction layers (threading, io) and to leverage functionality lower in the web services stack (WSRF, WSN, etc).

These components are architecturally diverse and it is thus hard to identify a overarching theme. Instead a few sub-themes have been identified and elaborated on in the below.

2. Conceptual Details

2.1. Web Services

We introduce basic concepts relating to Web services and their use and implementation within GT4, in particular within the "WS Core" (Java & C) components.

2.1.1. GT4, Distributed Systems, and Web Services

GT4 is a set of software components for building distributed systems: systems in which diverse and discrete software agents interact via message exchanges over a network to perform some tasks. Distributed systems face particular challenges relating to sometimes high and unpredictable network latencies, the possibility of partial failure, and issues of concurrency. In addition, system components may be located within distinct administrative domains, thus introducing issues of decentralized control and negotiation.

GT4 is, more specifically, a set of software components that (with some exceptions) implement Web services mechanisms for building distributed systems. Web services provide a standard means of interoperating between different software applications running on a variety of platforms and/or frameworks.

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

Web services standardize the messages that entities in a distributed system must exchange in order to perform various operations. At the lowest level, this standardization concerns the protocol used to transport messages (typically HTTP), message encoding (SOAP), and interface description (WSDL). A client interacts with a Web service by sending it a SOAP message; the client may subsequently receive response message(s) in reply. At higher levels, other specifications define conventions for securing message exchanges (e.g., WS-Security), for management (e.g., WSDM), and for higher-level functions such as discovery and choreography. Figure 1 presents a view of these different component technologies; we discuss specific specifications below in [Section 2.1.4, "Web Services Specifications"](#).

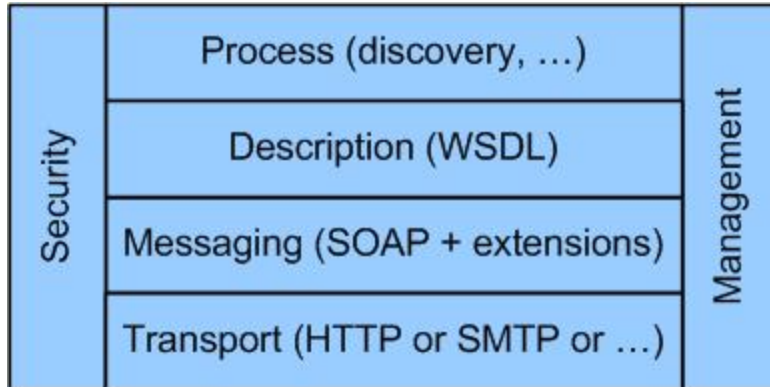


Figure 1: An abstract view of the various specifications that define the Web services architecture

2.1.2. Service Oriented Applications and Infrastructure

Web services technologies, and GT4 in particular, can be used to build both service-oriented applications and service-oriented infrastructure. Deferring discussion of the sometimes controversial term "service-oriented" to later in [Section 2.1.9, "Service Oriented Architecture"](#), we note that a service-oriented application is constructed via the composition of components defined by service interfaces (in the current context, Web services): for example, a financial or biological database, an options pricing routine, or a biological sequence analyzer. Many descriptions of Web services and SOAP focus on the task of defining interfaces to such components, often illustrating their discussion with examples such as a "stock quote service" (the "hello world" of Web services).

Particularly when servicing many such requests from a distributed community, we face the related problem of orchestrating and managing numerous distributed hardware and software components. Web services can be used for this purpose also, and thus we introduce the term service-oriented infrastructure to denote the resource management and provisioning mechanisms used to meet quality of service goals for components and applications. Many GT4 features are concerned with enabling the construction of service-oriented infrastructure.

2.1.3. Web Services Implementation

From the client perspective, a Web service is simply a network-accessible entity that processes SOAP messages. Things are somewhat more complex under the covers. To simplify service implementation, it is common for a Web services implementation to distinguish between:

1. the hosting environment (or container), the (domain-independent) logic used to receive a SOAP message and identify and invoke the appropriate code to handle the message, and potentially also to provide related administration functions, and:
2. the Web service implementation, the (domain-specific) code that handles the message.

This separation of concerns means that the developer need only provide the domain-specific message handling code to implement a new service. It is also common to further partition the hosting environment logic into that concerned with transporting the SOAP message (typically via HTTP, thus an "HTTP engine" or "Web server"-sometimes termed an "application server") and that concerned with processing SOAP messages (the "SOAP engine" or "SOAP processor"). Figure 2 illustrates these various components.

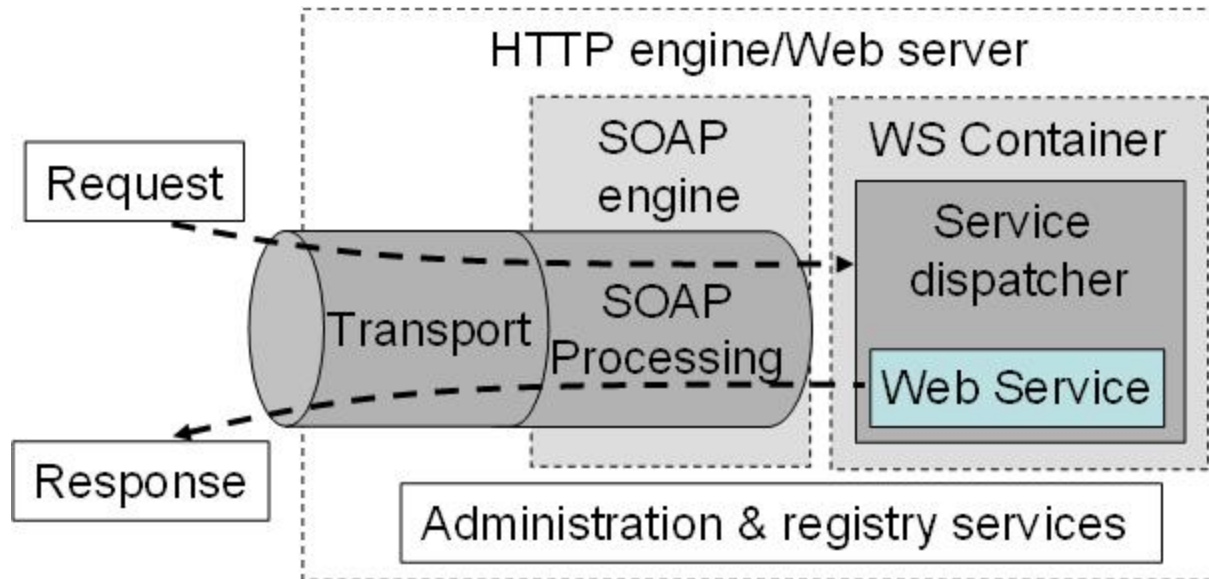


Figure 2: *WS Container*. High-level picture of functional components commonly encountered in Web service implementations, showing the path taken by requests and responses.

Many different containers exist, with different performance properties, supported Web services implementation languages, security support, and so forth. We mention below those used in GT4.

2.1.4. Web Services Specifications

We provide pointers to the Web services specifications that underlie GT4. These comprise the core specifications that define the Web services architecture (XML, SOAP, WSDL); WS-Security and other specifications relating to security; and the WS-Addressing, WSRF, and WS-Notification specifications used to define, name, and interact with stateful resources. We also speak briefly to emerging specifications that are likely to be important in future GT evolution. An important source of information on the requirements that motivate the use and development of these specifications is the Open Grid Services Architecture.

2.1.5. XML, SOAP, WSDL

XML is used extensively within Web services as a standard, flexible, and extensible data format. In addition to XML syntax, other important specifications are XML Schema and XML Namespaces. Note that while current Web services tools typically adopt a textual serialization, a binary encoding is also possible and may provide higher efficiency.

SOAP 1.2 provides a standard, extensible, composable framework for packaging and exchanging XML messages between a service provider and a service requester. SOAP is independent of the underlying transport protocol, but is most commonly carried on HTTP.

WSDL 1.1 is an XML document for describing Web services. Standardized binding conventions define how to use WSDL in conjunction with SOAP and other messaging substrates. WSDL interfaces can be compiled to generate proxy code that constructs messages and manages communications on behalf of the client application. The proxy automatically maps the XML message structures into native language objects that can be directly manipulated by the application. The proxy frees the developer from having to understand and manipulate XML.

2.1.6. WS-Security and Friends

The WS-Security family of specifications addresses a range of issues relating to authentication, authorization, policy representation, and trust negotiation in a Web services context. GT4 uses a number of these specifications plus other related specifications, notably Security Authorization Markup Language (SAML), to address message protection, authentication, delegation, and authorization, as follows:

- TLS (transport-level) or WS-Security and WS-SecureConversation (message level) are used as message protection mechanisms in combination with SOAP.
- X.509 End Entity Certificates or Username and Password are used as authentication credentials.
- X.509 Proxy Certificates and WS-Trust are used for delegation.
- SAML assertions are used for authorization.

2.1.7. WS-Addressing, WSRF, and WS-Notification

A number of related specifications provide functionality important for service oriented infrastructure in which we need to be able to represent and manipulate stateful entities such as physical resources of various kinds, logical components such as software licenses, and transient activities such as tasks and workflows.

The WS-Addressing specification defines transport-neutral mechanisms to address Web services and messages. Specifically, this specification defines XML elements to identify Web service endpoints and to secure end-to-end endpoint identification in messages.

The WS Resource Framework (WSRF) specifications define a generic and open framework for modeling and accessing stateful resources using Web services. This framework comprises mechanisms to describe views on the state (WS-ResourceProperties), to support management of the state through properties associated with the Web service (WS-ResourceLifetime), to describe how these mechanisms are extensible to groups of Web services (WS-ServiceGroup), and to deal with faults (WS-BaseFaults).

The WS-Notification family of specifications define a pattern-based approach to allowing Web services to disseminate information to one another. This framework comprises mechanisms for basic notification (WS-Notification), topic-based notification (WS-Topics), and brokered notification (WS-BrokeredNotification).

We note that the Web services standards space is in some turmoil due to competing proposed specifications. In particular, Microsoft and others recently proposed WS-Transfer, WS-Eventing, and WS-Management, which define similar functionality to WSRF, WS-Notification, and WSDM (discussed below), respectively, but using different syntax. We hope that these differences will be resolved in the future.

2.1.8. Other Relevant Specifications

The WS-Interoperability (WS-I) organization has produced a number of profiles that define ways in which existing Web services specifications can be used to promote interoperability among different implementations. The WS-I Basic Profile speaks to messaging and service description: primarily XML, SOAP, and WSDL. The WS-I Basic Security Profile speaks to basic security mechanisms. Other profiles are under development.

Web services distributed management (WSDM) specifications under development within OASIS are likely to play a role in future GT implementations as a means of managing GT components.

WS-CIM specifications under development within DMTF are likely to play a role in future GT implementations as a means of representing physical and virtual resources.

The Global Grid Forum's Open Grid Services Architecture (OGSA) working group has completed a document that provides a high-level description of the functionality required for future service-oriented infrastructure and applications, and a framework that suggests how this functionality can be factored into distinct specifications. The OGSA working group is now proceeding to define OGSA Profiles that, like WS-I profiles, will identify technical specifications that can be used to address specific Grid scenarios.

2.1.9. Service Oriented Architecture

We provide some additional discussion concerning the term service oriented architecture (SOA), which is used widely but not necessarily consistently within the Web services community. One common usage is simply to indicate the use of Web services technologies. However, the intention of those who coined the term seems to be rather to contrast two different styles of building distributed systems. Distributed object systems are distributed systems in which the semantics of object initialization and method invocation are exposed to remote systems by means of a proprietary or standardized mechanism to broker requests across system boundaries, marshal and unmarshal method argument data, etc. Distributed objects systems typically (albeit not necessarily) are characterized by objects maintaining a fairly complex internal state required to support their methods, a fine grained or "chatty" interaction between an object and a program using it, and a focus on a shared implementation type system and interface hierarchy between the object and the program that uses it.

In contrast, a Service Oriented Architecture (SOA) is a form of distributed systems architecture that is typically characterized by the following properties:

- Logical view: The service is an abstracted, logical view of actual programs, databases, business processes, etc., defined in terms of what it does, typically carrying out a business-level operation.
- Message orientation: The service is formally defined in terms of the messages exchanged between provider agents and requester agents, and not the properties of the agents themselves. The internal structure of an agent, including features such as its implementation language, process structure and even database structure, are deliberately abstracted away in the SOA: using the SOA discipline one does not and should not need to know how an agent implementing a service is constructed. A key benefit of this concerns so-called legacy systems. By avoiding any knowledge of the internal structure of an agent, one can incorporate any software component or application that can be "wrapped" in message handling code that allows it to adhere to the formal service definition.
- Description orientation: A service is described by machine-processable metadata. The description supports the public nature of the SOA: only those details that are exposed to the public and important for the use of the service should be included in the description. The semantics of a service should be documented, either directly or indirectly, by its description.
- Granularity: Services tend to use a small number of operations with relatively large and complex messages.
- Network orientation: Services tend to be oriented toward use over a network, though this is not an absolute requirement.
- Platform neutral: Messages are sent in a platform-neutral, standardized format delivered through the interfaces. XML is the most obvious format that meets this constraint.

It is argued that these features can allow service-oriented architectures to cope more effectively with issues that arise in distributed systems, such as problems introduced by latency and unreliability of the underlying transport, the lack of shared memory between the caller and object, problems introduced by partial failure scenarios, the challenges of concurrent access to remote resources, and the fragility of distributed systems if incompatible updates are introduced to any participant.

Web services technologies in general, and GT4 in particular, can be used to build both distributed object systems and service-oriented architectures. The specific design principles to be followed in a particular setting will depend on a variety of issues, including target environment, scale, platform heterogeneity, and expected future evolution.

3. Related Documents

3.1. Web Services

- Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C. and Orchard, D. Web Services Architecture. W3C, Working Draft¹

¹ <http://www.w3.org/TR/2003/WD-ws-arch-20030808/>

Chapter 2. GT 4.0 XIO: System Administrator's Guide

1. Introduction

Globus XIO is a development API. The only administration required is to install it according to the [GT 4.0 installation instructions](#)¹.

Important

This information is in addition to the basic Globus Toolkit prerequisite, overview, installation, security configuration instructions in the [GT 4.0 System Administrator's Guide](#)². Read through this guide before continuing!

2. Building and Installing

For instructions on building and installing GT 4.0, see the [Installation Guide](#)³.

3. Configuring

Globus XIO is a development API; therefore, configuration details are largely in the hands of the applications using it.

4. Deploying

Once a user has a successful installation of Globus XIO they may wish to use their own drivers. Once the driver is compiled, it can be used at runtime by Globus XIO so long as it can be found in the LD_LIBRARY_PATH.

5. Testing

A test suite is available to test the Globus XIO framework. To test it, simply run:

```
$GLOBUS_LOCATION/test/globus_xio_test/TESTS.pl
```

6. Security Considerations

Globus XIO is a framework for creating network protocols. Several existing protocols, such as TCP, come built into the framework. XIO itself introduces no known security risks. However, all network applications expose systems to the risks inherent when outsiders can connect to them. Also included in the XIO distribution is the gsi driver, which provides a driver that allows for secure connections.

¹ <http://www.globus.org/toolkit/docs/4.0/admin/docbook/ch04.html>

² <http://www.globus.org/toolkit/docs/4.0/admin/docbook/>

³ <http://www.globus.org/toolkit/docs/4.0/admin/docbook/ch04.html>

7. Troubleshooting

- The environment variable `GLOBUS_LOCATION` must be set to a valid Globus 4.0 installation.
- Various other environment variables must be set in order to easily use the GlobusXIO application. The proper environment can be established by running: `source $GLOBUS_LOCATION/etc/globus-user-env.sh` or `source $GLOBUS_LOCATION/etc/globus-user-env.csh` depending on the shell you are using.

Chapter 3. GT 4.0 XIO : User's Guide

1. Introduction

Globus XIO is a developers API. There are no user level tools.

2. Command-line tools

XIO is strictly a library; there is no command line tool for XIO.

3. Graphical user interfaces

Globus XIO has no GUI.

4. Troubleshooting

Please see the [Troubleshooting info in the Developer's Guide](#).

Chapter 4. GT 4.0 XIO : Developer's Guide

1. Introduction

This guide contains information of interest to developers working with XIO. It provides reference information for application developers, including APIs, architecture, procedures for using the APIs and code samples.

2. Before you begin

2.1. Feature summary

Features new in release 4.0

- UDT driver.
- Mode E Driver
- Telnet Driver
- Queuing Driver
- Ordering Driver
- Dynamically loadable drivers.

Other Supported Features

- Single API to swappable IO implementations.
- Asynchronous IO support.
- Native timeout support.
- Data descriptors for providing driver specific hints.
- Modular driver stacks to maximize code reuse.
- TCP, UDP, file, HTTP, telnet, mode E, GSI drivers.

Deprecated Features

- GSSAPI_FTP driver now distributed with the GridFTP Server

2.2. Tested platforms

Tested Platforms for XIO:

- Linux
 - Mandrakelinux release 10.1

- SuSE Linux 9.1 (i586)
- Debian GNU/Linux 3.1
- Red Hat Linux release 9
- SunOS
 - SunOS 5.9 sun4u sparc SUNW,Sun-Fire-280R
- MacOS
 - Darwin Kernel Version 7.9.0
- Additionally all platforms supported by GT4.0 [Platforms](#)¹

2.3. Backward compatibility summary

Protocol changes since GT version 3.2

- None.

API changes since GT version 3.2

- `globus_xio_stack_copy` added to the API. This allows a user to duplicated a configured stack.
- `globus_xio_driver_set_eof_received` added to the driver API. This function allows drivers to have multiple outstanding reads at one time.
- `globus_xio_driver_eof_received` added to the driver API. Working in conjunction with `globus_xio_driver_set_eof_received` to allow drivers to have multiple outstanding reads.
- Users can now pass in a NULL callback for timeouts and it is assumed that when time expires the user wants the operation to timeout. Previously a user callback was required where the user would decide if they wanted the timeout.

2.4. Technology dependencies

XIO depends on the following GT components:

- Globus Core
- Globus Common
- Globus GSSAPI

2.5. Security considerations

Globus XIO is a framework for creating network protocols. Several existing protocols, such as TCP, come built into the framework. XIO itself introduces no known security risks. However, all network applications expose systems to the risks inherent when outsiders can connect to them. Also included in the XIO distribution is the `gsi` driver, which provides a driver that allows for secure connections.

¹ <http://www.globus.org/toolkit/docs/4.0/admin/docbook/ch03.html#s-platform>

3. Architecture and design overview

3.1. Introduction

This document shall explain the external view of the Globus XIO architecture. Globus XIO is broken down into two main components, framework and drivers. The following picture illustrates the architecture:

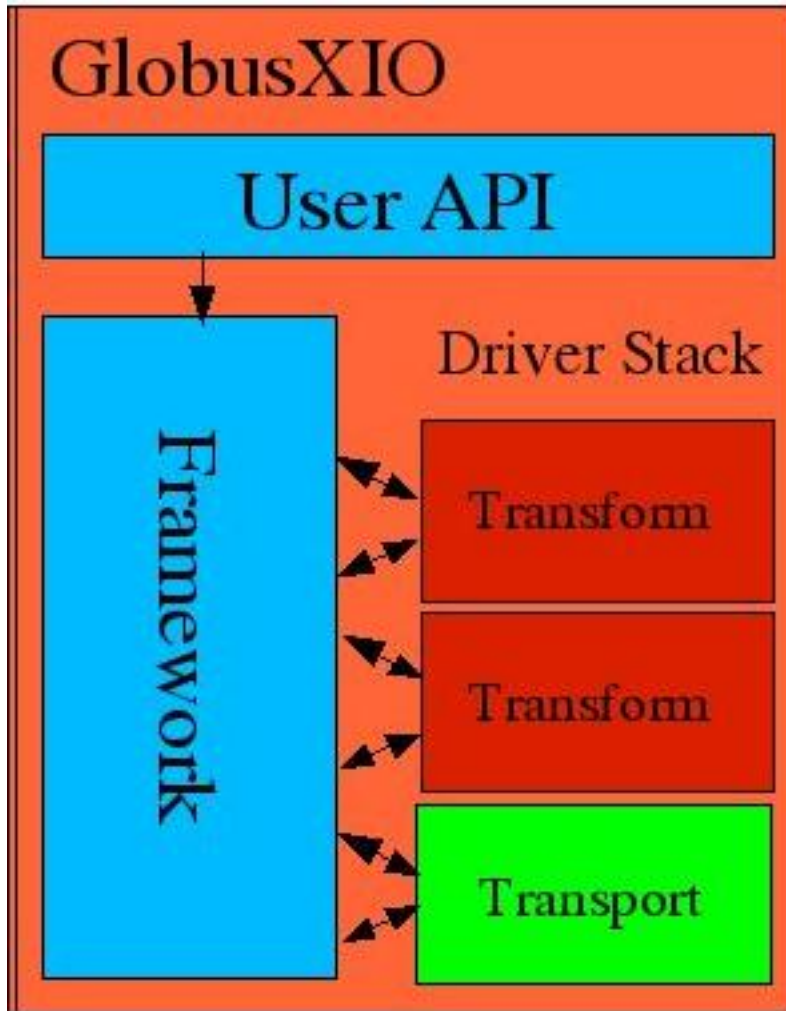


Figure 1

3.2. Framework

The Globus XIO framework manages IO operation requests that an application makes via the user API. The framework does no work to deliver the data in an IO operation nor does it manipulate the data. All of that work is done by the drivers. The framework's job is to manage requests and map them to the drivers interface. It is the drivers themselves that are responsible for manipulating and transporting the data.

3.3. Drivers

A driver is the component of Globus XIO that is responsible for manipulating and transporting the users data. There are two types of drivers, transform and transport. Transform drivers are those that manipulate the data buffers passed to it via the user API and the XIO framework. Transport drivers are those that are capable of sending the data over a wire.

Drivers are grouped into stacks, that is, one driver on top of another. When an IO operation is requested the Globus XIO framework passes the operation request to every driver in the order they are in the stack. When the bottom level driver (the transport driver) finishes shipping the data, it passes the data request back to the XIO framework. Globus XIO will then deliver the request back up the stack in this manner until it reaches the top, at which point the application will be notified that there request is completed.

In a driver stack there can only be one transport driver. The reason for this is that the transport driver is the one responsible for sending or receiving the data. Once this type of operation is performed it makes no sense to pass the request down the stack, as the data has just been transfered. It is now time to pass the operation back up the stack.

There can be many transform drivers in any given driver stack. A transform driver is one that can manipulate the requested operations as they pass. Some good examples of transform drivers are security wrappers and compression. However, a transport driver can also be one that adds additional protocol. For example a stack could consist of a TCP transport driver and an HTTP transform driver. The HTTP driver would be responsible for marshaling the HTTP protocol and the TCP driver would be responsible for shipping that protocol over the wire.

3.4. Example

In the following picture we illustrate a user application using Globus XIO to speak the GridFTP protocol across a TCP connection:

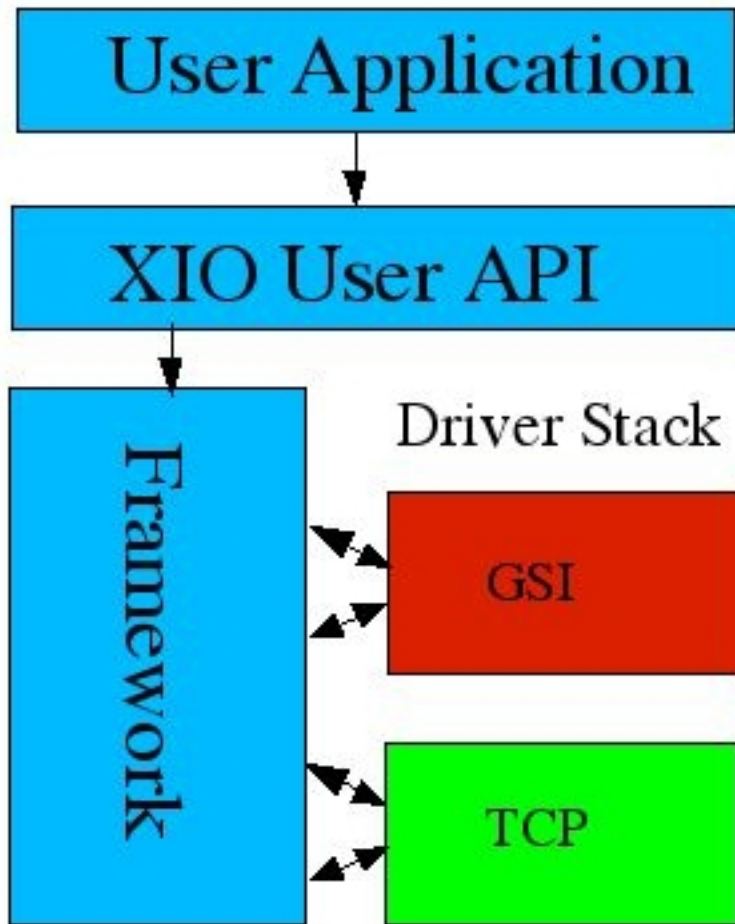


Figure 2

The user has built a stack consisting of one transform driver and one transport driver. TCP is the transport driver in this stack, and as all transport modules must be, it is at the bottom of the stack. Above TCP is the GSI transform driver which performs necessary messaging to authenticate a user and the integrity of the data.

The first thing the user application will do after building the stack is call the XIO user API function `globus_xio_open()`. The Globus XIO framework will create internal data structures for tracking this operation and then pass the operation request to the GSI driver. The GSI driver has nothing to do before the underlying stack has opened a handle so it simply passes the request down the stack. The request is thereby passed to the TCP driver. The TCP driver will then execute the socket level transport code contained within it to establish a connection to the given contact string.

Once the TCP connection has been established the TCP driver will notify the XIO framework that it has completed its request and thereby the GSI driver will be notified that the open operation it had previously passed down the stack has now completed. At this point the GSI driver will start the authentication processes (note that at this point the user does not yet have an open handle). The GSI driver has an open handle and upon it several sends and receives are performed to authenticate the connection. If the GSI driver is not satisfied with the authentication process it closes the handle it has to the stack below it and tells the XIO framework that it has completed the open request with an error. If it is satisfied it simply tells the XIO framework that it has completed the open operation. The user is now notified that the open operation completed, and if it was successful they now have an open handle.

Other operations work in much the same way. When a user posts a read the read request is first delivered to the GSI driver. The GSI driver will wrap the buffer and pass the modified buffer down the stack. The framework will then de-

liver the write request with the newly modified buffer to the TCP driver. The TCP driver will write the data across the socket mapped to this handle. When it finishes it notifies the framework, which notifies the GSI driver. The GSI driver has nothing more to do so it notifies the framework that it is complete and the framework then notifies the user.

3.5. Driver Interface

There is a well defined interface to a driver. Drivers are modular components with specific tasks. The purpose of drivers in the Globus XIO library is extensibility. As more and more protocols are developed, more and more drivers can be written to implement these protocols. As new drivers are written they can be added to Globus XIO as either statically linked libraries or dynamically loaded libraries. In the case of dynamic loading it is not even necessary to recompile existing source code. Each driver has a unique name according to the Globus XIO driver naming convention. A program simply needs to be aware of this name (this can be passed in via the command line) and the Globus XIO framework will be responsible for loading the driver.



Note

The above example is simplified for the purposes of understanding. There are optimizations built into Globus XIO which alter the course of events outlined above. However, conceptually the above is accurate.

4. Public interface

The documentation for the public interfaces can be found at: [API Doc](#)².

5. Usage scenarios

5.1. Using the Globus XIO API for IO operations within C programs

5.1.1. Introduction

This Guide explains how to use the Globus XIO API for IO operations within C programs. Since Globus XIO is a simple API it is pretty straight forward. The best way to become familiar with it is by looking at an example. See [globus_xio_example.c](#).

5.1.2. Activate Globus

Let's examine the case where a user wishes to use Globus XIO for reading data from a file. As in all globus programs the first thing that must be done is to activate the globus module. Until activated no globus_xio function calls can be successfully executed. It is activated with the following line:

```
globus_module_activate(GLOBUS_XIO_MODULE);
```

5.1.3. Load Driver

Let's examine the case where a user wishes to use Globus XIO for reading data. The next step is to load all the drivers needed to complete the IO operations in which you are interested. The function globus_xio_load_driver() is used to load a driver. In order to successfully call this function you must know the name of all the drivers you wish to load.

² http://www.globus.org/api/c/globus_xio/html/index.html

For this example we only want to load the file IO driver. The prepackaged file IO driver's name is: "file". This driver would be loaded with the following code:

```
globus_result_t          res;  
globus_xio_driver_t     driver;  
  
res = globus_xio_driver_load(&driver, "file");
```

If upon completion of the above function call `res` returns `GLOBUS_SUCCESS` then the driver was successfully loaded and can be referenced with the variable "driver".

5.1.4. Create Stack

Now that `globus_xio` is activated and we have a driver loaded we need to build a driver stack. In our example the stack is very simple as it consists of only one driver, the file driver. The stack is established with the following code (building off of the above code snippets):

```
globus_xio_stack_t      stack;  
  
globus_xio_stack_init(&stack);  
globus_xio_stack_push_driver(stack, driver);
```

5.1.5. Opening Handle

Now that the stack is created we can open a handle to the file. There are two ways that a handle can be opened. The first is a passive open. An example of this is a TCP listener. The open is performed passively by waiting for some other entity to act upon it. The second is an active open. An active open is the opposite of a passive open. The TCP counter example for this is a connect. The users initiates the open. In our example we shall be performing an active open.

Before opening a handle it must be initialized. The following illustrates initialization for client side handles:

```
globus_xio_handle_t     handle;  
  
res = globus_xio_handle_create(&handle, stack);
```

Server side handles are a bit more complicated. First we must introduce the data structure `globus_xio_server_t`. This structure shares many concepts with a TCP listener, mainly that it spawns handles ("connections") as passive open requests are made. If the user wishes to accept a new connection a call to `globus_xio_accept()` or `globus_xio_register_accept()` will initialize a new handle:

```
globus_xio_server_t     server;  
globus_xio_handle_t     handle;  
globus_result_t         res;  
  
res = globus_xio_server_create(&server_handle, NULL, stack);  
res = globus_xio_server_accept(&handle, server);
```

Once the handle is initialized it should be opened in order to perform read and write operations upon it. If the handle is a client then a "contact string" is required. This string represents the target that the user wishes to open:

```
globus_xio_attr_t      attr;
char *                contact_string = "file:/etc/groups";

globus_xio_attr_init(&attr);
globus_xio_open(xio_handle, contact_string, attr);
globus_xio_attr_destroy(attr);
```

note: attrs can be used to color the behaviors of a handle. For a conceptual understanding, they are not important and a user is free to simply pass NULL wherever an attr is required.

Now that we have an open handle to a file we can read or write data to it with either `globus_xio_read()` or `globus_xio_write()`. Once we are finished performing IO operations on the handle `globus_xio_close(handle)` should be called.

5.1.6. Pay Off

This may seem like quite a bit of effort for simply reading a file, and it is. However the advantages become clear when exploring the swapping of other drivers. In the above example it would be trivial to change the IO operations from file IO to TCP, or HTTP, or ftp. All the user would need to do is change the driver name string passed to `globus_xio_load_driver()` and the contact string passed to `globus_xio_target_init()`. This can easily be done at runtime, as the program [globus_xio_example.c](#)³ demonstrates.

So the little program [globus_xio_example.c](#)⁴ has the ability to be any reading client or server (HTTP, ftp, TCP, file, etc) as long as the proper drivers are in the LD_LIBRARY_PATH.

5.2. Driver Quick Start

5.2.1. Introduction

This Guide explains how to create a transport driver for Globus XIO. For the purpose of exploring both what a transform driver is and how to write one this guide will walk through an example driver. The full source code for the driver can be found at [Driver Example](#)⁵. This example implements a file driver for `globus_xio`. If a user of `globus_xio` were to put this file at the bottom of the stack, they could access files on the local file system.

5.2.2. Data Structures

There are three data structures that will be explored in this example: attribute, target, and handle. The driver defines the memory layout of these data structures but the `globus_xio` framework imposes certain semantics upon them. It is up to the driver how to use them, but `globus_xio` will be expecting certain behaviors.

5.2.3. Attributes

Each driver may have its own attribute structure. The attribute gives the `globus_xio` user API an opportunity to tweak parameters inside the driver. The single attribute structure is used for all possible driver specific attributes:

1. Target attributes

³ [globus_xio_example.c](#)

⁴ [globus_xio_example.c](#)

⁵ [globus_xio_driver_example.c](#)

2. Handle attributes

3. Server attributes

How each of these can use the attribute structure will be unveiled as the tutorial continues. For now it is simply important to remember there is attribute structure used to initiate of the driver ADTs.

A driver is not required to have an attribute support at all. However if the driver author chooses to support attributes the following functions must be implemented:

```
typedef globus_result_t
(*globus_xio_driver_attr_init_t)(
    void **                                     out_attr);

typedef globus_result_t
(*globus_xio_driver_attr_cntl_t)(
    void *                                     attr,
    int                                        cmd,
    va_list                                    ap);

typedef globus_result_t
(*globus_xio_driver_attr_copy_t)(
    void **                                     dst,
    void *                                     src);

typedef globus_result_t
(*globus_xio_driver_attr_destroy_t)(
    void *                                     attr);
```

See [driver API doc](#)⁶ for more information.

We shall now take our first look at the file [Driver Example](#)⁷. The file driver needs a way to provide the user level programmer with a means of setting the mode and flags when a file is open (akin to the POSIX function `open()`).

The first step in creating this ability is to a) define the attribute structure and b) implement the `globus_xio_driver_attr_init_t` function which will initialize it:

```
/*
 * attribute structure
 */
struct globus_l_xio_file_attr_s
{
    int                                     mode;
    int                                     flags;
}

globus_result_t
globus_xio_driver_file_attr_init(
    void **                                     out_attr)
{
    struct globus_l_xio_file_attr_s *        file_attr;
```

⁶ http://www.globus.org/api/c-globus-4.0/globus_xio_gridftp_driver/html/index.html

⁷ `globus_xio_driver_example.c`

```

/*
 * create a file attr structure and initialize its values
 */
file_attr = (struct globus_l_xio_file_attr_s *)
    globus_malloc(sizeof(struct globus_l_xio_file_attr_s));

file_attr->flags = O_CREAT;
file_attr->mode = S_IRWXU;

/* set the out parameter to the driver attr */
*out_attr = file_attr;

return GLOBUS_SUCCESS;
}

```

The above simply defines a structure that can hold two integers, mode and flags, then defines a function that will allocate and initialize this structure. `globus_xio` hides much of the memory management of these attribute structures from the driver. However, it does need the driver to provide a means of copying them, and free all resources associated with them. In the case of the file driver example, these are both simple:

```

globus_result_t
globus_xio_driver_file_attr_copy(
    void **                                dst,
    void *                                  src)
{
    struct globus_l_xio_file_attr_s *      file_attr;

    file_attr = (struct globus_l_xio_file_attr_s *)
        globus_malloc(sizeof(struct globus_l_xio_file_attr_s));

    memcpy(file_attr, src, sizeof(struct globus_l_xio_file_attr_s));

    *dst = file_attr;

    return GLOBUS_SUCCESS;
}

globus_result_t
globus_xio_driver_file_attr_destroy(
    void *                                  attr)
{
    globus_free(attr);

    return GLOBUS_SUCCESS;
}

```

The above code should be fairly clear. Obviously we need a method with which the user can set flags and mode. This is accomplished with the following interface function:

```

globus_result_t
globus_xio_driver_file_attr_cntl(
    void *
    int
    va_list
                                attr,
                                cmd,
                                ap)

```

```

{
    struct globus_l_xio_file_attr_s *      file_attr;
    int *                                  out_i;

    file_attr = (struct globus_l_xio_file_attr_s *)attr;
    switch(cmd)
    {
        case GLOBUS_XIO_FILE_SET_MODE:
            file_attr->mode = va_arg(ap, int);
            break;

        case GLOBUS_XIO_FILE_GET_MODE:
            out_i = va_arg(ap, int *);
            *out_i = file_attr->mode;
            break;

        case GLOBUS_XIO_FILE_SET_FLAGS:
            file_attr->flags = va_arg(ap, int);
            break;

        case GLOBUS_XIO_FILE_GET_FLAGS:
            out_i = va_arg(ap, int *);
            *out_i = file_attr->flags;
            break;

        default:
            return FILE_DRIVER_ERROR_COMMAND_NOT_FOUND;
            break;
    }

    return GLOBUS_SUCCESS;
}

```

This function is called passing the driver an initialized `file_attr` structure, a command, and a variable argument list.

Based on the value of `cmd`, the driver decides to do one of the following:

- set flags or mode from the `va_args`
- return flags or mode to the user via a pointer in `va_args`

5.2.4. Target

A target structure represents what a driver will open. It is initialized from a contact string and an attribute. In the case of a file driver, the target simply holds onto the contact string as a path to the file.

The file driver implements the following target functions:

```

globus_result_t
globus_xio_driver_file_target_init(
    void **                                  out_targ
    void *                                  target_
    const char *                             contact_strin

```

```

        globus_xio_driver_stack_t          stack)
{
    struct globus_l_xio_file_target_s *    target;

    /* create the target structure and copy the contact string into it */
    target = (struct globus_l_xio_file_target_s *)
        globus_malloc(sizeof(struct globus_l_xio_file_target_s));
    strncpy(target->pathname, contact_string, sizeof(target->pathname) - 1);
    target->pathname[sizeof(target->pathname) - 1] = '\\0';

    return GLOBUS_SUCCESS;
}

/*
 * destroy the target structure
 */
globus_result_t
globus_xio_driver_file_target_destroy(
    void *                                target)
{
    globus_free(target);

    return GLOBUS_SUCCESS;
}

```

The above function handles the creation and destruction of the file driver's target structure.



Note

When the target is created, the contact string is copied into it. It is invalid to just copy the pointer to the contact string. As soon as this interface function returns, that pointer is no longer valid.

5.2.5. Handle

The most interesting of the three data types discussed here is the handle. All typical IO operations (open, close, read, write) are performed on the handle. The handle is the initialized form of the target and an attr. The driver developer should use this ADT to keep track of any state information they will need in order to perform reads and writes.

In the example case, the driver handle is fairly simple as the driver is merely a wrapper around POSIX calls:

```

struct globus_l_xio_file_handle_s
{
    int                fd;
};

```

The reader should review the following functions in [Driver Example](#)⁸ in order to see how the handle structure is used:

- globus_xio_driver_file_open()
- globus_xio_driver_file_write()
- globus_xio_driver_file_read()

⁸ globus_xio_driver_example.c

- `globus_xio_driver_file_close()`

5.2.6. IO Operations

The read and write interface functions are called in response to a user read or write request. Both functions are provided with a vector that has at least the same members as the `struct iovec` and a vector length. As of now, the `iovec` elements may contain extra members, so if you wish to use `readv()` or `writew()`, you will have to transfer the `iov_base` and `iov_len` members to the POSIX `iovec`.

As with the open and close interface functions, if an error occurs before any real processing has occurred, the interface function may simply return the error (in a `result_t`), effectively canceling the operation. However, once bytes have been read or written, you must not return the error. You must report the number of bytes read/written along with the result.

When an operation is done, either by error or successful completion, the operation must be 'finished'. To do this, a call must be made to:

```
globus_result_t
globus_xio_driver_finished_read/write(
    globus_xio_driver_operation_t      op,
    globus_result_t                    res,
    globus_ssize_t                     nbytes);
```

5.2.6.1. Blocking vs Non-blocking Calls

In general, the driver developer does not need to concern himself with how the user made the call. Whether it was a blocking or an asynchronous call, XIO will handle things correctly.

However the call was made, the driver developer can call `globus_xio_driver_finished_{open, read, write, close}` either while in the original interface call, in a separate thread, or in a separate callback kick out via the `globus_callback` API.

5.2.7. The Glue

Through a process not finalized yet, XIO will request the `globus_xio_driver_t` structure from the driver. This structure defines all of the interface functions that the driver supports. In detail:

```
/*
 * main io interface functions
 */
globus_xio_driver_open_t           open_func;
globus_xio_driver_close_t         close_func;
globus_xio_driver_read_t          read_func;
globus_xio_driver_write_t         write_func;
globus_xio_driver_handle_cntl_t   handle_cntl_func;

globus_xio_driver_target_init_t    target_init_func;
globus_xio_driver_target_destroy_t target_destroy_func;

/*
 * target init functions. Must have client or server
 */
globus_xio_driver_server_init_t    server_init_func;
globus_xio_driver_server_accept_t  server_accept_func;
globus_xio_driver_server_destroy_t server_destroy_func;
```

```

globus_xio_driver_server_cntl_t                                server_cntl_func

/*
 * driver attr functions. All or none may be NULL
 */
globus_xio_driver_attr_init_t                                attr_init_f
globus_xio_driver_attr_copy_t                                attr_copy_func
globus_xio_driver_attr_cntl_t                                attr_cntl_fu
globus_xio_driver_attr_destroy_t                             attr_destroy_fun

/*
 * data descriptor functions. All or none
 */
globus_xio_driver_data_descriptor_init_t                     dd_init;
globus_xio_driver_driver_data_descriptor_copy_t              dd_copy;
globus_xio_driver_driver_data_descriptor_destroy_t           dd_destroy;
globus_xio_driver_driver_data_descriptor_cntl_t              dd_cntl;

```

6. Tutorials

- [PowerPoint Tutorial](#)⁹

7. Debugging

All standard C debugging techniques apply to debugging XIO applications. Also, Globus XIO provides users with some additional debugging information. If the environment variable GLOBUS_XIO_DEBUG is set debugging information will be written to a file or stdout. The information generated is particularly useful to identify a suspect bug in Globus XIO. GLOBUS_XIO_DEBUG is set in the following way:

```
GLOBUS_XIO_DEBUG="<level> [,[[#]<file name>][,<flag>[,<timestamp_levels>]]"
```

The value of <level> can take on the logical OR of any of the following:

- GLOBUS_XIO_DEBUG_ERROR = 1
- GLOBUS_XIO_DEBUG_WARNING = 2
- GLOBUS_XIO_DEBUG_TRACE = 4
- GLOBUS_XIO_DEBUG_INTERNAL_TRACE = 8
- GLOBUS_XIO_DEBUG_INFO = 16
- GLOBUS_XIO_DEBUG_STATE = 32
- GLOBUS_XIO_DEBUG_INFO_VERBOSE = 64

<file name> is a debug output file, if empty stderr will be used by default. If a '#' precedes the filename, the file will be overwritten on each run. Otherwise, the output will be appended to the existing file.

<flags>

⁹ xio.ppt

- 0 = default
- 1 = show thread ids
- 2 = append the pid to debug filename

8. Troubleshooting

- The environment variable `GLOBUS_LOCATION` must be set to a valid Globus 4.0 installation.
- Various other environment variables must be set in order to easily use the Globus XIO application. The proper environment can be established by running: `source $GLOBUS_LOCATION/etc/globus-user-env.sh` or `source $GLOBUS_LOCATION/etc/globus-user-env.csh`, depending on the shell you are using.

9. Related Documentation

- [Performance](#)¹⁰
- [Previous HTML doc](#)¹¹

¹⁰ http://www-unix.mcs.anl.gov/~kettimut/xio/XIO_Performance.pdf

¹¹ <http://www-unix.mcs.anl.gov/~bresnaha/xio>

Chapter 5. GT 4.0 Component Fact Sheet: XIO

1. Brief component overview

Globus XIO is an extensible input/output library written in C for the Globus Toolkit. It provides a single API (open/close/read/write) that supports multiple wire protocols, with protocol implementations encapsulated as drivers. The XIO drivers distributed with 4.0 include TCP, UDP, file, HTTP, GSI, GSSAPI_FTP, TELNET and queuing.

In addition, Globus XIO provides a driver development interface for use by protocol developers. This interface allows the developer to concentrate on writing protocol code rather than infrastructure, as XIO provides a framework for error handling, asynchronous message delivery, timeouts, etc.

The XIO driver-based approach maximizes the reuse of code by supporting the notion of a driver stack. XIO drivers can be written as atomic units and stacked on top of one another. This modularization provides maximum flexibility and simplifies the design and evaluation of individual protocols.

2. Summary of features

Features new in release 4.0

- UDT driver.
- Mode E Driver
- Telnet Driver
- Queuing Driver
- Ordering Driver
- Dynamically loadable drivers.

Other Supported Features

- Single API to swappable IO implementations.
- Asynchronous IO support.
- Native timeout support.
- Data descriptors for providing driver specific hints.
- Modular driver stacks to maximize code reuse.
- TCP, UDP, file, HTTP, telnet, mode E, GSI drivers.

Deprecated Features

- GSSAPI_FTP driver now distributed with the GridFTP Server

3. Usability summary

The API for GlobusXIO has not changed to any significant degree since its introduction. All other usability issues relate to the compilation and installation of the libraries and are therefore subject to the usability of the installation tools found at the [Toolkit Downloads](http://www.globus.org/toolkit/downloads/)¹ page.

4. Backward compatibility summary

Protocol changes since GT version 3.2

- None.

API changes since GT version 3.2

- `globus_xio_stack_copy` added to the API. This allows a user to duplicated a configured stack.
- `globus_xio_driver_set_eof_received` added to the driver API. This function allows drivers to have multiple outstanding reads at one time.
- `globus_xio_driver_eof_received` added to the driver API. Working in conjunction with `globus_xio_driver_set_eof_received` to allow drivers to have multiple outstanding reads.
- Users can now pass in a NULL callback for timeouts and it is assumed that when time expires the user wants the operation to timeout. Previously a user callback was required where the user would decide if they wanted the timeout.

5. Technology dependencies

XIO depends on the following GT components:

- Globus Core
- Globus Common
- Globus GSSAPI

6. Tested platforms

Tested Platforms for XIO:

- Linux
 - Mandrakelinux release 10.1
 - SuSE Linux 9.1 (i586)
 - Debian GNU/Linux 3.1
 - Red Hat Linux release 9
- SunOS

¹ <http://www.globus.org/toolkit/downloads/>

- SunOS 5.9 sun4u sparc SUNW,Sun-Fire-280R
- MacOS
 - Darwin Kernel Version 7.9.0
- Additionally all platforms supported by GT4.0 [Platforms](#)²

7. Associated standards

Associated standards for XIO:

Adoption of standards in XIO is determined by the drivers. All drivers distributed by XIO are compliant with the protocol they implement.

8. For more information

Click [here](#)³ for more information about this component.

² <http://www.globus.org/toolkit/docs/4.0/admin/docbook/ch03.html#s-platform>

³ [index.html](#)

Chapter 6. GT 4.0 Component Guide to Public Interfaces: XIO

1. Semantics and Syntax of XIO APIs

1.1. Programming Model Overview

Globus XIO is based on an event based asynchronous programming model. This model is described in great detail at: [Asynchronous Event Handling](#)¹. In short, with Globus XIO, connections are opened and closed. While open, read and write requests are posted with a callback function pointer given by the user. When the event completes, the given callback is called.

1.2. Component API

You can find documentation of the XIO library at:

http://www.globus.org/api/c/globus_xio/html/index.html

For information on the internationalization API, see the [Common Libraries Public Interface](#)².

2. Semantics and Syntax of the WSDL

There is no WSDL associated with Globus XIO.

3. Command-Line Tools

XIO is strictly a library; there is no command line tool for XIO.

4. Overview of Graphical User Interface

Globus XIO has no GUI.

5. Semantics and Syntax of Domain-Specific Interface

This category of interfaces does not apply to XIO.

6. Configuration Interface

Globus XIO is a development API; therefore, configuration details are largely in the hands of the applications using it.

¹ [../globus-async.html](#)

² http://www.globus.org/toolkit/docs/4.0/common/ccommonlib/C_Common_Libraries_Public_Interfaces.html#apis

7. Environment Variable Interface

The vast majority of the environment variables that effect the Globus XIO framework are defined by the driver in use. The following are links to descriptions of the more common driver environment variables:

- http://www.globus.org/api/c-globus-4.0/globus_xio/html/group_tcp_driver_envs.html
- http://www.globus.org/api/c-globus-4.0/globus_xio/html/group_file_driver_envs.html
- http://www.globus.org/api/c-globus-4.0/globus_xio/html/group_gsi_driver_envs.html
- http://www.globus.org/api/c-globus-4.0/globus_xio/html/group_udp_driver_envs.html

Chapter 7. GT 4.0 XIO: Quality Profile

1. Test coverage reports

- <http://www.globus.org/xio/coverage/>

2. Code analysis reports

- <http://www.globus.org/xio/coverage/>

3. Outstanding bugs

- [Bug 3082](#)¹
- [Bug 3083](#)²
- [Bug 3086](#)³
- [Bug 2219](#)⁴

4. Bug Fixes

- [Bug 1416](#)⁵
- [Bug 1851](#)⁶
- [Bug 3028](#)⁷
- [Bug 2602](#)⁸
- [Bug 2348](#)⁹
- [Bug 3061](#)¹⁰
- [Bug 1707](#)¹¹

5. Performance reports

- http://www-unix.mcs.anl.gov/~kettimut/xio/XIO_Performance.pdf

¹ http://bugzilla.globus.org/globus/show_bug.cgi?id=3082

² http://bugzilla.globus.org/globus/show_bug.cgi?id=3083

³ http://bugzilla.globus.org/globus/show_bug.cgi?id=3086

⁴ http://bugzilla.globus.org/globus/show_bug.cgi?id=2219

⁵ http://bugzilla.globus.org/globus/show_bug.cgi?id=1416

⁶ http://bugzilla.globus.org/globus/show_bug.cgi?id=1851

⁷ http://bugzilla.globus.org/globus/show_bug.cgi?id=3028

⁸ http://bugzilla.globus.org/globus/show_bug.cgi?id=2602

⁹ http://bugzilla.globus.org/globus/show_bug.cgi?id=2348

¹⁰ http://bugzilla.globus.org/globus/show_bug.cgi?id=3061

¹¹ http://bugzilla.globus.org/globus/show_bug.cgi?id=1707

Chapter 8. GT 4.0 Migrating Guide for XIO

The following provides information about migrating from previous versions of the Globus Toolkit.

1. Migrating from GT2

Globus XIO is a new component as of Globus 3.2. Previous versions of Globus provided an API called `globus_io`. XIO is a replacement for `globus_io`; however a compatibility layer exists to allow full backward compatibility for applications written against `globus_io`. For users familiar with `globus_io` the learning curve for XIO will be very small. The APIs use the same event model and only significantly differ in connection establishment.

2. Migrating from GT3

Globus XIO has made very few changes in the API since its introduction. User of GT4.0 who are previously familiar with `globus_xio` will notice no difference.

Chapter 9. GT 4.0.8 Incremental Release Notes: XIO

1. Introduction

These release notes are for the incremental release 4.0.8. It includes a summary of changes since 4.0.7, bug fixes since 4.0.7 and any known problems that still exist at the time of the 4.0.8 release. This page is in addition to the top-level 4.0.8 release notes at <http://www.globus.org/toolkit/releasenotes/4.0.8>.

For release notes about 4.0 (including feature summary, technology dependencies, etc) go to the [XIO 4.0 Release Notes](#)¹.

2. Changes Summary

Added randomness to the TCP port selection. When GLOBUS_TCP_SOURCE_RANGE is set, instead of starting from the lower bound and monotonically increasing to find the first open port, a random number in the specified range will be picked. see [Bug 6185](#)² for more details.

3. Bug Fixes

- [Bug 6185](#)³ Adding randomness to the TCP port selection

4. Known Problems

- [Bug 2530](#)⁴ END_OF_ENTITY
- [Bug 3147](#)⁵ XIO doesn't provide for multiplexed drivers passing down driver_handle_cntl
- [Bug 4186](#)⁶ Some GRAM security errors less informative than before
- [Bug 4354](#)⁷ Encryption may cause a deadlock.
- [Bug 4429](#)⁸ Memory leaks when using GSI driver from XIO library.

5. For More Information

Click [here](#)⁹ for more information about this component.

¹ http://www.globus.org/toolkit/docs/4.0/common/xio/XIO_Release_Notes.html

² http://bugzilla.globus.org/globus/show_bug.cgi?id=6185

³ http://bugzilla.globus.org/globus/show_bug.cgi?id=6185

⁴ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2530

⁵ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3147

⁶ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4186

⁷ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4354

⁸ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4429

⁹ [index.html](#)

Chapter 10. GT 4.0.7 Incremental Release Notes: XIO

1. Introduction

These release notes are for the incremental release 4.0.7. It includes a summary of changes since 4.0.6, bug fixes since 4.0.6 and any known problems that still exist at the time of the 4.0.7 release. This page is in addition to the top-level 4.0.7 release notes at <http://www.globus.org/toolkit/releasenotes/4.0.7>.

For release notes about 4.0 (including feature summary, technology dependencies, etc) go to the [XIO 4.0 Release Notes](#)¹.

2. Changes Summary

Other than bug fixes, no changes have been made for XIO since 4.0.6.

3. Bug Fixes

- [Bug 5004](#):² A race in deactivation of globus_common caused a seg fault. globus_mutex_lock() was being called on a destroyed mutex. This is solved now.
- [Bug 5503](#):³ UDP driver sendto() fails on Solaris

4. Known Problems

- [Bug 2530](#):⁴ END_OF_ENTITY
- [Bug 3147](#):⁵ XIO doesn't provide for multiplexed drivers passing down driver_handle_cntl
- [Bug 4186](#):⁶ Some GRAM security errors less informative than before
- [Bug 4354](#):⁷ Encryption may cause a deadlock.
- [Bug 4429](#):⁸ Memory leaks when using GSI driver from XIO library.

5. For More Information

Click [here](#)⁹ for more information about this component.

¹ http://www.globus.org/toolkit/docs/4.0/common/xio/XIO_Release_Notes.html

² http://bugzilla.globus.org/globus/show_bug.cgi?id=5004

³ http://bugzilla.globus.org/globus/show_bug.cgi?id=5503

⁴ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2530

⁵ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3147

⁶ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4186

⁷ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4354

⁸ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4429

⁹ [index.html](#)

Chapter 11. GT 4.0.6 Incremental Release Notes: XIO

1. Introduction

These release notes are for the incremental release 4.0.6. It includes a summary of changes since 4.0.5, bug fixes since 4.0.5 and any known problems that still exist at the time of the 4.0.6 release. This page is in addition to the top-level 4.0.6 release notes at <http://www.globus.org/toolkit/releasenotes/4.0.6>.

For release notes about 4.0 (including feature summary, technology dependencies, etc) go to the [XIO 4.0 Release Notes](#)¹.

2. Changes Summary

No changes have been made for XIO in the 4.0 branch since 4.0.5.

3. Bug Fixes

No bugs have fixed since the 4.0.5 release.

4. Known Problems

- [Bug 2530](#):² END_OF_ENTITY
- [Bug 3147](#):³ XIO doesn't provide for multiplexed drivers passing down driver_handle_cntl
- [Bug 4186](#):⁴ Some GRAM security errors less informative than before
- [Bug 4354](#):⁵ Encryption may cause a deadlock.
- [Bug 4429](#):⁶ Memory leaks when using GSI driver from XIO library.
- [Bug 5004](#):⁷ Failed assert in globus_l_xio_system_unregister_write on x86_64 platform

5. For More Information

Click [here](#)⁸ for more information about this component.

¹ http://www.globus.org/toolkit/docs/4.0/common/xio/XIO_Release_Notes.html

² http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2530

³ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3147

⁴ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4186

⁵ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4354

⁶ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4429

⁷ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=5004

⁸ [index.html](#)

Chapter 12. GT 4.0.5 Incremental Release Notes: XIO

1. Introduction

These release notes are for the incremental release 4.0.5. It includes a summary of changes since 4.0.4, bug fixes since 4.0.4 and any known problems that still exist at the time of the 4.0.5 release. This page is in addition to the top-level 4.0.5 release notes at <http://www.globus.org/toolkit/releasenotes/4.0.5>.

For release notes about 4.0 (including feature summary, technology dependencies, etc) go to the [XIO 4.0 Release Notes](#)¹.

2. Changes Summary

No changes have been made for XIO since 4.0.4.

3. Bug Fixes

No bugs have fixed since the 4.0.4 release.

4. Known Problems

- [Bug 2219](#):² xio_close in an xio callback hangs
- [Bug 2530](#):³ END_OF_ENTITY
- [Bug 3082](#):⁴ context leak
- [Bug 3147](#):⁵ XIO doesn't provide for multiplexed drivers passing down driver_handle_cntl
- [Bug 4186](#):⁶ Some GRAM security errors less informative than before
- [Bug 4354](#):⁷ Encryption may cause a deadlock.
- [Bug 4429](#):⁸ Memory leaks when using GSI driver from XIO library.
- [Bug 4624](#):⁹ globus_xio_close() dumps a core
- [Bug 4995](#):¹⁰ occasional hanging in threaded XIO

¹ http://www.globus.org/toolkit/docs/4.0/common/xio/XIO_Release_Notes.html

² http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2219

³ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2530

⁴ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3082

⁵ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3147

⁶ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4186

⁷ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4354

⁸ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4429

⁹ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4624

¹⁰ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4995

- [Bug 5004](#):¹¹ Failed assert in globus_l_xio_system_unregister_write on x86_64 platform

5. For More Information

Click [here](#)¹² for more information about this component.

¹¹ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=5004

¹² [index.html](#)

Chapter 13. GT 4.0.4 Incremental Release Notes: XIO

1. Introduction

These release notes are for the incremental release 4.0.4. It includes a summary of changes since 4.0.3, bug fixes since 4.0.3 and any known problems that still exist at the time of the 4.0.4 release. This page is in addition to the top-level 4.0.4 release notes at <http://www.globus.org/toolkit/releasenotes/4.0.4>.

For release notes about 4.0 (including feature summary, technology dependencies, etc) go to the [XIO 4.0 Release Notes](#)¹.

2. Changes Summary

No changes have been made for XIO since 4.0.3 other than the listed bugfixes.

3. Bug Fixes

- [Bug xxx](#):² describe

4. Known Problems

- [Bug 2219](#):³ xio_close in an xio callback hangs
- [Bug 2530](#):⁴ END_OF_ENTITY
- [Bug 3082](#):⁵ context leak
- [Bug 3147](#):⁶ XIO doesn't provide for multiplexed drivers passing down driver_handle_cntl
- [Bug 4186](#):⁷ Some GRAM security errors less informative than before
- [Bug 4354](#):⁸ Encryption may cause a deadlock.
- [Bug 4429](#):⁹ Memory leaks when using GSI driver from XIO library.
- [Bug 4624](#):¹⁰ globus_xio_close() dumps a core
- [Bug 4995](#):¹¹ occasional hanging in threaded XIO

¹ http://www.globus.org/toolkit/docs/4.0/common/xio/XIO_Release_Notes.html

² http://bugzilla.globus.org/globus/show_bug.cgi?id=xxx

³ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2219

⁴ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2530

⁵ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3082

⁶ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3147

⁷ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4186

⁸ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4354

⁹ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4429

¹⁰ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4624

¹¹ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4995

- [Bug 5004](#):¹² Failed assert in globus_l_xio_system_unregister_write on x86_64 platform

5. For More Information

Click [here](#)¹³ for more information about this component.

¹² http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=5004

¹³ [index.html](#)

Chapter 14. GT 4.0.3 Incremental Release Notes: XIO

1. Introduction

These release notes are for the incremental release 4.0.3. It includes a summary of changes since 4.0.2, bug fixes since 4.0.2 and any known problems that still exist at the time of the 4.0.3 release. This page is in addition to the top-level 4.0.3 release notes at <http://www.globus.org/toolkit/releasenotes/4.0.3>.

For release notes about 4.0 (including feature summary, technology dependencies, etc) go to the [XIO 4.0 Release Notes](#)¹.

2. Changes Summary

There were no changes to this release of XIO.

3. Bug Fixes

No bugs were fixed for XIO since 4.0.2.

4. Known Problems

The following problems are known to exist for XIO at the time of the 4.0.4 release:

- [Bug 3082](#).² Under certain circumstances there is a small memory leak.

5. For More Information

Click [here](#)³ for more information about this component.

¹ http://www.globus.org/toolkit/docs/4.0/common/xio/XIO_Release_Notes.html

² http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3082

³ [index.html](#)

Chapter 15. GT 4.0.2 Incremental Release Notes: XIO

1. Introduction

These release notes are for the incremental release 4.0.2. It includes a summary of changes since 4.0.1, bug fixes since 4.0.1 and any known problems that still exist at the time of the 4.0.2 release. This page is in addition to the top-level 4.0.2 release notes at <http://www.globus.org/toolkit/releasenotes/4.0.2>.

For release notes about 4.0 (including feature summary, technology dependencies, etc) go to the [XIO 4.0 Release Notes](#)¹.

2. Changes Summary

Other than minor bug fixes, there were no changes to this release of XIO.

3. Bug Fixes

The following bug was fixed for XIO:

- [Bug 3733](#):² Potential buffer overflow in http driver

4. Known Problems

The following problems are known to exist for XIO at the time of the 4.0.2 release:

- [Bug 3082](#):³ Under certain circumstances there is a small memory leak.

5. For More Information

Click [here](#)⁴ for more information about this component.

¹ http://www.globus.org/toolkit/docs/4.0/common/xio/XIO_Release_Notes.html

² http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3733

³ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3082

⁴ [index.html](#)

Chapter 16. GT 4.0.1 Incremental Release Notes: XIO

1. Introduction

These release notes are for the incremental release 4.0.1. It includes a summary of changes since 4.0.0, bug fixes since 4.0.0 and any known problems that still exist at the time of the 4.0.1 release. This page is in addition to the top-level 4.0.1 release notes at <http://www.globus.org/toolkit/releasenotes/4.0.1>.

For release notes about 4.0 (including feature summary, technology dependencies, etc) go to the [XIO 4.0 Release Notes](#)¹.

2. Changes Summary

Other than a bug fix, there were no changes for XIO.

3. Bug Fixes

The following bug was fixed for XIO:

- [Bug 3410](#):² Reporting an error on writew on some systems

4. Known Problems

There are no known problems with XIO at the time of the 4.0.1 release.

5. For More Information

Click [here](#)³ for more information about this component.

¹ http://www.globus.org/toolkit/docs/4.0/common/xio/XIO_Release_Notes.html

² http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3410

³ [index.html](#)

Chapter 17. GT 4.0 Release Notes: XIO

1. Component Overview

Globus XIO is an extensible input/output library written in C for the Globus Toolkit. It provides a single API (open/close/read/write) that supports multiple wire protocols, with protocol implementations encapsulated as drivers. The XIO drivers distributed with 4.0 include TCP, UDP, file, HTTP, GSI, GSSAPI_FTP, TELNET and queuing.

In addition, Globus XIO provides a driver development interface for use by protocol developers. This interface allows the developer to concentrate on writing protocol code rather than infrastructure, as XIO provides a framework for error handling, asynchronous message delivery, timeouts, etc.

The XIO driver-based approach maximizes the reuse of code by supporting the notion of a driver stack. XIO drivers can be written as atomic units and stacked on top of one another. This modularization provides maximum flexibility and simplifies the design and evaluation of individual protocols.

2. Feature Summary

Features new in release 4.0

- UDT driver.
- Mode E Driver
- Telnet Driver
- Queuing Driver
- Ordering Driver
- Dynamically loadable drivers.

Other Supported Features

- Single API to swappable IO implementations.
- Asynchronous IO support.
- Native timeout support.
- Data descriptors for providing driver specific hints.
- Modular driver stacks to maximize code reuse.
- TCP, UDP, file, HTTP, telnet, mode E, GSI drivers.

Deprecated Features

- GSSAPI_FTP driver now distributed with the GridFTP Server

3. Bug Fixes

- [Bug 1416](#)¹
- [Bug 1851](#)²
- [Bug 3028](#)³
- [Bug 2602](#)⁴
- [Bug 2348](#)⁵
- [Bug 3061](#)⁶
- [Bug 1707](#)⁷

4. Known Problems

- [Bug 3082](#)⁸
- [Bug 3083](#)⁹
- [Bug 3086](#)¹⁰
- [Bug 2219](#)¹¹

5. Technology Dependencies

XIO depends on the following GT components:

- Globus Core
- Globus Common
- Globus GSSAPI

6. Tested Platforms

Tested Platforms for XIO:

- Linux
 - Mandrakelinux release 10.1

¹ http://bugzilla.globus.org/globus/show_bug.cgi?id=1416

² http://bugzilla.globus.org/globus/show_bug.cgi?id=1851

³ http://bugzilla.globus.org/globus/show_bug.cgi?id=3028

⁴ http://bugzilla.globus.org/globus/show_bug.cgi?id=2602

⁵ http://bugzilla.globus.org/globus/show_bug.cgi?id=2348

⁶ http://bugzilla.globus.org/globus/show_bug.cgi?id=3061

⁷ http://bugzilla.globus.org/globus/show_bug.cgi?id=1707

⁸ http://bugzilla.globus.org/globus/show_bug.cgi?id=3082

⁹ http://bugzilla.globus.org/globus/show_bug.cgi?id=3083

¹⁰ http://bugzilla.globus.org/globus/show_bug.cgi?id=3086

¹¹ http://bugzilla.globus.org/globus/show_bug.cgi?id=2219

- SuSE Linux 9.1 (i586)
- Debian GNU/Linux 3.1
- Red Hat Linux release 9
- SunOS
 - SunOS 5.9 sun4u sparc SUNW,Sun-Fire-280R
- MacOS
 - Darwin Kernel Version 7.9.0
- Additionally all platforms supported by GT4.0 [Platforms](#)¹²

7. Backward Compatibility Summary

Protocol changes since GT version 3.2

- None.

API changes since GT version 3.2

- `globus_xio_stack_copy` added to the API. This allows a user to duplicated a configured stack.
- `globus_xio_driver_set_eof_received` added to the driver API. This function allows drivers to have multiple outstanding reads at one time.
- `globus_xio_driver_eof_received` added to the driver API. Working in conjunction with `globus_xio_driver_set_eof_received` to allow drivers to have multiple outstanding reads.
- Users can now pass in a NULL callback for timeouts and it is assumed that when time expires the user wants the operation to timeout. Previously a user callback was required where the user would decide if they wanted the timeout.

8. For More Information

Click [here](#)¹³ for more information about this component.

¹² <http://www.globus.org/toolkit/docs/4.0/admin/docbook/ch03.html#s-platform>

¹³ <http://www.globus.org/toolkit/docs/4.0/common/xio/>