

GT 4.2.0 Java WS A&A Admin Guide

DRAFT

GT 4.2.0 Java WS A&A Admin Guide

Introduction

This guide contains advanced configuration information for system administrators working with Java WS A&A. It provides references to information on procedures typically performed by system administrators, including installing, configuring, deploying, and testing the installation.

Important

This information is in addition to the basic Globus Toolkit prerequisite, overview, installation, security configuration instructions in the [Installing GT 4.2.0](#). Read through this guide before continuing!

Authentication/message protection issues. The main administration issues for authentication/message-protection deal with configuring credential-related settings. There are multiple mechanisms for doing this:

- Security Descriptors (This is the *preferred* mechanism)
 - Container Security Descriptor
 - Service Security Descriptor
- CoG properties
- Environment variables
- Relying on default behavior. The only default behaviors available concern the proxy file and trusted certificates locations.

Authorization issues. Authorization in Java WS A&A is enforced on the server and the client side. Admin configuration could include determining the container/service level authorization mechanism and setting up and managing authorization policy, e.g. entries in *grid map file* and so on. The [Chapter 2, Configuring](#) chapter describes how to configure Java WS A&A descriptors.

Table of Contents

- 1. Building and Installing 1
- 2. Configuring 2
 - 1. Configuring authorization 2
 - 2. Configuring authentication/message protection 3
- 3. Deploying 4
- 4. Testing 5
- 5. Security Considerations 8
 - 1. Security considerations for Java WS A&A 8
- 6. Debugging 9
 - 1. Logging in Java WS Core 9
- 7. Troubleshooting 11
 - 1. Credential Troubleshooting 11
 - 2. Error Messages For Java WS A&A 14
 - 3. For more troubleshooting 16
- Glossary 17

DRAFT

List of Tables

2.1. Configuring server side authentication and message/transport security	3
7.1. Credential Errors	12
7.2. Java WS A&A Errors	15

DRAFT

Chapter 1. Building and Installing

This component is built and installed as a part of Java WS Core. See "Building and Installing" in the [Java WS Core Admin Guide](#) for more information.

DRAFT

Chapter 2. Configuring

Java WS A&A is configured using security descriptors. The following describes configuration settings specific for authorization and authentication. You can read the entire Java WS A&A Security Descriptor documentation [here](#).

- [Configuring authorization](#)
- [Configuring authentication/message protection](#)

1. Configuring authorization

1.1. Configuration overview

Security descriptors are mechanisms used to configure authorization mechanism and policy. The authorization on the server side can be configured at the container, service or resource level.

On the client side, authorization can be configured using security descriptors or as a property on the stub. This configuration can be done on a per invocation granularity

1.2. Server side authorization

The server side authorization can be configured at the container, service or resource level using

- Security descriptors using files. Refer to [Section 1, “Configuring Using Files”](#)
- Security descriptors programmatically. Refer to [Section 2, “Configuring Programmatically ”](#)

To write and configure a server-side custom authorization mechanism refer to [Section 2.3, “Writing a custom server-side authorization mechanism”](#).

1.3. Client side authorization

The client side authorization can be configured for each invocation.

- Security descriptors using files. Refer to [Section 1, “Configuring Using Files”](#), specifically [Section 1.2.2, “Configuring authorization mechanism ”](#).
- Security descriptors programmatically. Refer to [Section 2, “Configuring Programmatically ”](#)
- Properties on the Stub. Refer to [Section 2.1, “Configuring client-side authorization on the stub”](#)

To write and configure custom authorization mechanism refer to [Section 2.2, “Writing custom client-side authorization scheme”](#).

If no authorization mechanism has been specified, HostOrSelf authorization is used. In this scheme host authorization is tried first, if it fails, self authorization is attempted

2. Configuring authentication/message protection

2.1. Configuration overview

Configuration of service-side security settings can be achieved by using container or service security descriptor. Some of the security configuration, like the credential to use and trusted certificates location, can also be configured using CoG properties or rely on default location. **The preferred way is to provide these settings in a security descriptor.**

The next section provides details on the relevant properties. An overview of the syntax of security descriptors can be found in [Java WS A&A Security Descriptor Framework](#). Available CoG security properties can be found in [Chapter 2, Configuring](#)

2.2. Syntax of the interface

The following properties are relevant to authentication and message/transport security:

Table 2.1. Configuring server side authentication and message/transport security

Number	Task	Descriptor Configuration	Alternate Configuration
1	Credentials	Container or service descriptor configuration	<ul style="list-style-type: none"> • X509_USER_CERT or CoG Configuration: User certificate configuration • X509_USER_KEY or CoG Configuration: User key configuration • X509_USER_PROXY or CoG Configuration: User proxy configuration <p>If no explicit configuration is found, the default proxy is read from /tmp/x509_up_<uid>.</p>
2	Trusted Certificates	Container security descriptor configuration	CoG Configuration
3	Limited proxy policy configuration	Container or service descriptor configuration	None.
4	Replay Attack Window	Container or service descriptor configuration	None.
5	Replay Attack Filter	Container or service descriptor configuration	None.
6	Replay timer interval	Container descriptor configuration	None.
7	Context timer interval	Container descriptor configuration	None.

Chapter 3. Deploying

This component is deployed as a part of [Java WS Core](#).

DRAFT

Chapter 4. Testing

To execute security tests ensure that [Ant with JUnit is configured](#).

All the security tests require a valid credential. Refer to [Security section of GT Administrator guide](#) for details on acquiring credentials.

The security tests are included in `$GLOBUS_LOCATION/lib/wsrf_test_unit.jar`. This jar contains tests for both the Java WS Core component and the WS Authentication and Authorization components contained in the Java WS Core package.

To execute the tests, pass the above jar file to the test script as described in [Section 12.3, “How do I run my JUnit tests for Java WS Core and/or my services?”](#). To ensure that only security tests are run, set `-DsecurityTestsOnly=true`.

By default the tests require that the container and the tests use the same credentials, i.e self authorization is done on secure calls.

The tests allow for another configuration in which the container can be configured with [host credentials](#) and the tests can be run with any credentials.

- Configure the container to use host credentials using the security descriptor as described in the [container descriptor](#) section.
- Edit `$GLOBUS_LOCATION/etc/globus_wsrf_test_unit/server-config.wsdd`.
 - Comment out the configured descriptor in `SecurityTestService`, `RPPParamTestService` and `AuthzCalloutTestService` that specifies self authorization.

```
<!-- Does self authz by default -->
<!-- parameter name="securityDescriptor"
      value="@config.dir@/security-config.xml" / -->
```

- Uncomment the configuration for identity authorization.

```
<!-- For use only when identity authz is used-->
<parameter name="securityDescriptor"
      value="@config.dir@/identity-security-config.xml" />
```

- In `$GLOBUS_LOCATION/etc/globus_wsrf_test_unit/identity-security-config.xml`, set the value of the property `identity` to the subject DN of the credentials used to run the tests.

```
<!-- set to expected identity -->
<param:parameter name="identity"
      value="Identity used by client" />
```

- Edit `$GLOBUS_LOCATION/etc/globus_wsrf_test_unit/test-container-security-config.xml` to add the following element, before `<reject-limited-proxy value="true"/>`

```

<credential>
  <cert-key-files>
    <key-file value="path to container key"/>
    <cert-file value="path to container cert"/>
  </cert-key-files>
</credential>

```

- Start a secure and insecure standalone container.

```

$ cd $GLOBUS_LOCATION
$ bin/globus-start-container -nosec

```

On another window,

```

$ cd $GLOBUS_LOCATION
$ bin/globus-start-container

```

- To run tests against external containers, secure and insecure, on localhost ports 8180 and 8181 respectively, the command would be:

```

ant -f share/globus_wsrf_test/runtests.xml runServer
-Dtests.jar=$GLOBUS_LOCATION/lib/wsrf_test_unit.jar
-DsecurityTestsOnly=true
-Djunit.jvmarg=-Dsecurity.test.client.authz=host
-Dsecurity.test.authz.identity="container/suject/DN"
-Dsecurity.test.enc.cred="/server/public/certificate"
-Dsecurity.test.server.cert="/server/public/certificate/file"
-Dsecurity.test.server.key="/server/key/file"
-Dtest.server.url=http://127.0.0.1:8181/wsrf/services/
-Dsecure.test.server.url=https://127.0.0.1:8180/wsrf/services/

```

- For example, if the container security descriptor has credentials configured as */etc/grid-security/containercert.pem* and */etc/grid-security/containerkey.pem* and the DN of the credential is */DC=org/OU=Services/CN=-testDN/CN=some.host.edu*, the command to run would be:

```

ant -f share/globus_wsrf_test/runtests.xml runServer
-Dtests.jar=$GLOBUS_LOCATION/lib/wsrf_test_unit.jar
-DsecurityTestsOnly=true
-Djunit.jvmarg=-Dsecurity.test.client.authz=host
-Dtest.server.url=http://127.0.0.1:8181/wsrf/services/
-Dsecure.test.server.url=https://127.0.0.1:8180/wsrf/services/
-Dsecurity.test.authz.identity="/DC=org/OU=Services/CN=-testDN/CN=some.host.edu"
-Dsecurity.test.server.cert="/etc/grid-security/containercert.pem"

```

```
-Dsecurity.test.server.key="/etc/grid-security/containerkey.pem"  
-Dsecurity.test.enc.cred="/etc/grid-security/containercert.pem"
```

DRAFT

Chapter 5. Security Considerations

1. Security considerations for Java WS A&A

1.1. Security considerations for authorization

1.1.1. Client side authorization

Client authorization of the server is done after the completion of the operation when GSI Secure Message authentication is used. If you require client authorization to be done prior, use GSI Secure Conversation or GSI *Transport security*.

1.1.2. Host authorization

During host authorization, the toolkit treats DNs "hostname-*.edu" as equivalent to "hostname.edu". This means that if a service was setup to do host authorization and hence accept the certificate "hostname.edu", it would also accept certificates with DNs "hostname-*.edu".

The feature is in place to allow a multi-homed host following a "hostname-interface" naming convention, to have a single host certificate. For example, host "grid.test.edu" would also accept likes of "grid-1.test.edu" or "grid-foo.test.edu".

Note

The wildcard character "*" matches only name of the host and not domain components. This means that "hostname.edu" will not match "hostname-foo.sub.edu", but will match "host-foo.edu".

Note

If a host was set up to accept "hostname-1.edu", it will not accept any of "hostname-*.edu".

A [bug](#)¹ has been opened to see if this feature needs to be modified.

1.2. Security considerations for Message/Transport-level Security

1.2.1. File permissions

The Java security code currently does not enforce secure permissions and, implicitly, file ownership requirements on any of the security related files, e.g. configuration and credential files. It is thus important that administrators ensure that the relevant files have correct permissions and ownership. Permissions should generally be as restrictive as possible, i.e. *private keys* should be readable only by the file owner and other files should be writable by owner only, and the files should generally be owned by the globus user (the requirements that the C code enforces are documented in [Configuring GSI](#)).

Also refer to [Section 5, "Known Problems"](#) for details on any other open issues.

¹ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2969

Chapter 6. Debugging

Because Java WS A&A is built on Java WS Core, it uses the same sys admin logging, described below:

1. Logging in Java WS Core

The following information applies to Java WS Core and all services built on Java WS Core.

Java WS Core server side has two types of loggers. One logger is used for development logging and by default writes to standard out. The other logger includes system administration information and is [CEDPs best practices](#)¹ compliant.

On client side, only developer logging is available and is configured using `log4j.properties`.

1.1. Development Logging in Java WS Core

The following information applies to Java WS Core and those services built on it.

Logging in the Java WS Core is based on the [Jakarta Commons Logging](#)² API. Commons Logging provides a consistent interface for instrumenting source code while at the same time allowing the user to plug-in a different logging implementation. Currently we use [Log4j](#)³ as a logging implementation. Log4j uses a separate configuration file to configure itself. Please see Log4j documentation for details on the [configuration file format](#)⁴.

1.1.1. Configuring server side developer logs

Server side logging can be configured in `$GLOBUS_LOCATION/container-log4j.properties`, when the container is stand alone container. For tomcat level logging, refer to [Logging for Tomcat](#)⁵. The logger `log4j.appender.A1` is used for developer logging and by default writes output to the system output. By default it is set for all warnings in the Globus Toolkit package to be displayed.

Additional logging can be enabled for a package by adding a new line to the configuration file. Example:

```
#for debug level logging from org.globus.package.FooClass
log4j.category.org.globus.package.name.FooClass=DEBUG
#for warnings from org.some.warn.package
log4j.category.org.some.warn.package=WARN
```

1.1.2. Configuring client side developer logs

Client side logging can be configured in `$GLOBUS_LOCATION/log4j.properties`. The logger `log4j.appender.A1` is used for developer logging and by default writes output to the system output. By default it is set for all warnings in the Globus Toolkit package to be displayed.

¹ <http://cedps.net/index.php/LoggingBestPractices>

² <http://jakarta.apache.org/commons/logging/>

³ <http://logging.apache.org/log4j/>

⁴ [http://logging.apache.org/log4j/docs/api/org/apache/log4j/PropertyConfigurator.html#doConfigure\(java.lang.String,org.apache.log4j.spi.LoggerRepository\)](http://logging.apache.org/log4j/docs/api/org/apache/log4j/PropertyConfigurator.html#doConfigure(java.lang.String,org.apache.log4j.spi.LoggerRepository))

⁵ <http://tomcat.apache.org/tomcat-5.5-doc/logging.html>

1.2. Configuring system administration logs

The specific logger to edit will be `log4j.logger.sysadmin` in `$GLOBUS_LOCATION/container-log4j.properties`. There you can configure the following properties:

```
log4j.appender.infoCategory=org.apache.log4j.RollingFileAppender
log4j.appender.infoCategory.Threshold=INFO
log4j.appender.infoCategory.File=var/containerLog
log4j.appender.infoCategory.MaxFileSize=10MB
log4j.appender.infoCategory.MaxBackupIndex=2
```

Above implies the logging file is rolling with each file size limited to 10MB and the logging information is stored in `$GLOBUS_LOCATION/var/containerLog`.

1.3. Sample log file

The [sample log file](#)⁶ contains many log entries for various scenarios in the Java WS container.

⁶ <http://www.globus.org/toolkit/docs/4.2/4.2.0/common/javawscore/sample-container-log.txt>

Chapter 7. Troubleshooting

For a list of common errors in GT, see [Error Codes](#).

1. Credential Troubleshooting

1.1. Credential Errors

The following are some common problems that may cause clients or servers to report that credentials are invalid:

For a list of common errors in GT, see [Error Codes](#).

DRAFT

Table 7.1. Credential Errors

Error Code	Definition	Possible Solutions
Your proxy credential may have expired	Your proxy credential may have expired.	Use <code>grid-proxy-info</code> to check whether the proxy credential has actually expired. If it has, generate a new proxy with <code>grid-proxy-init</code> .
The system clock on either the local or remote system is wrong.	This may cause the server or client to conclude that a credential has expired.	Check the system clocks on the local and remote system.
Your end-user certificate may have expired	Your end-user certificate may have expired	Use <code>grid-cert-info</code> to check your certificate's expiration date. If it has expired, follow your CA's procedures to get a new one.
The permissions may be wrong on your proxy file	If the permissions on your proxy file are too lax (for example, if others can read your proxy file), Globus Toolkit clients will not use that file to authenticate.	You can "fix" this problem by changing the permissions on the file or by destroying it (with <code>grid-proxy-destroy</code>) and creating a new one (with <code>grid-proxy-init</code>). Important: However, it is still possible that someone else has made a copy of that file during the time that the permissions were wrong. In that case, they will be able to impersonate you until the proxy file expires or your permissions or end-user certificate are revoked, whichever happens first.
The permissions may be wrong on your private key file	If the permissions on your end user certificate private key file are too lax (for example, if others can read the file), <code>grid-proxy-init</code> will refuse to create a proxy certificate.	You can "fix" this by changing the permissions on the private key file. Important: However, you will still have a much more serious problem: it is possible that someone has made a copy of your private key file. Although this file is encrypted, it is possible that someone will be able to decrypt the private key, at which point they will be able to impersonate you as long as your end user certificate is valid. You should contact your CA to have your end-user certificate revoked and get a new one.
The remote system may not trust your CA	The remote system may not trust your CA	Verify that the remote system is configured to trust the CA that issued your end-entity certificate. See Installing GT 4.2.0 for details.
You may not trust the remote system's CA	You may not trust the remote system's CA	Verify that your system is configured to trust the remote CA (or that your environment is set up to trust the remote CA). See Installing GT 4.2.0 for details.
There may be something wrong with the remote service's credentials	There may be something wrong with the remote service's credentials	It is sometimes difficult to distinguish between errors reported by the remote service regarding your credentials and errors reported by the client interface regarding the remote service's credentials. If you cannot find anything wrong with your credentials, check for the same conditions on the remote system (or ask a remote administrator to do so).

1.2. Some tools to validate certificate setup

1.2.1. Check that the user certificate is valid

```
openssl verify -CApath /etc/grid-security/certificates
-purpose sslclient ~/.globus/usercert.pem
```

1.2.2. Connect to the server using s_client

```
openssl s_client -ssl3 -cert ~/.globus/usercert.pem -key
~/.globus/userkey.pem -CApath /etc/grid-security/certificates
-connect <host:port>
```

Here `<host:port>` denotes the server and port you connect to.

If it prints an error and puts you back at the command prompt, then it typically means that the *server* has closed the connection, i.e. that the server was not happy with the client's certificate and verification. Check the SSL log on the server.

If the command "hangs" then it has actually opened a telnet style (but secure) socket, and you can "talk" to the server.

You should be able to scroll up and see the subject names of the server's verification chain:

```
depth=2 /DC=net/DC=ES/O=ESnet/OU=Certificate Authorities/CN=ESnet Root CA 1
verify return:1
depth=1 /DC=org/DC=DOEGrids/OU=Certificate Authorities/CN=DOEGrids CA 1
verify return:1
depth=0 /DC=org/DC=doegrids/OU=Services/CN=wiggum.mcs.anl.gov
verify return:1
```

In this case, there were no errors. Errors would give you an extra line next to the subject name of the certificate that caused the error.

1.2.3. Check that the server certificate is valid

Requires root login on server:

```
openssl verify -CApath /etc/grid-security/certificates -purpose sslserver
/etc/grid-security/hostcert.pem
```

2. Error Messages For Java WS A&A

DRAFT

Table 7.2. Java WS A&A Errors

Error Code	Definition	Possible Solutions
[JWSSEC-248] Secure container requires valid credentials	This error occurs when <code>globus-start-container</code> is run without any valid credentials. Either a proxy certificate or service/host certificate needs to be configured for the container to start up.	<ol style="list-style-type: none"> <li data-bbox="805 300 1334 604">1. If you are not looking to start up a container that uses GSI Secure Transport, which is used by the container by default, use <code>globus-start-container -nosec</code>. You will be able to use insecure clients and services. However, this also implies that if you have not configured individual services with credentials, you will not be able to securely access the service. <li data-bbox="805 615 1334 804">2. If you are running a personal container, generate a proxy certificate with <code>grid-proxy-init</code>. If the proxy certificate is not in the default location, configure the container security descriptor as described in Configuring Container Security Descriptor. <li data-bbox="805 825 1334 930">3. If you want to use host certificates, configure the container security descriptor as described Configuring Credentials.
Failed to start container: Container failed to initialize [Caused by: [JWSSEC-250] Failed to load certificate/key file]	This error occurs if the file path to the container certificate and key configured are invalid.	<ol style="list-style-type: none"> <li data-bbox="805 963 1334 1163">1. The path to the container certificate and key are configured in <code>\$GLOBUS_LOCATION/etc/globus_wsrf_core/global_security_descriptor.xml</code>. This file is loaded as described [here - fixme link]. Ensure that the path is correct.
Failed to start container: Container failed to initialize [Caused by: [JWSSEC-249] Failed to load proxy file]	This error occurs if container proxy file configured is invalid.	<ol style="list-style-type: none"> <li data-bbox="805 1194 1334 1394">1. The path to the container proxy certificates are configured in <code>\$GLOBUS_LOCATION/etc/globus_wsrf_core/global_security_descriptor.xml</code>. This file is loaded as described [here - fixme link]. Ensure that the path is correct.
Failed to start container: Container failed to initialize [Caused by: [JWSSEC-245] Error parsing file: "etc/globus_wsrf_core/global_security_descriptor.xml" [Caused by: ...]	This error occurs if the container security descriptor configured is invalid.	<ol style="list-style-type: none"> <li data-bbox="805 1425 1334 1520">1. The container security descriptor should conform to the Container Security Descriptor Schema.¹ <li data-bbox="805 1541 1334 1604">2. Refer to the "Caused by: " section for details on the specific element that is not correct.

¹ http://www.globus.org/toolkit/docs/4.2.0/security/container_security_descriptor.xsd

Error Code	Definition	Possible Solutions
[JGLOBUS-77] Unknown CA	This error occurs if the CA certificate for the credentials being used is not installed correctly.	<ol style="list-style-type: none">1. If this issue occurs on the server side, the container is not configured with CA certificates. The container looks for trusted certificates in the default location as described Java CoG Toolkit FAQ²2. On the server side, the trusted certificates can be configured as described in Trusted Certificates3. On the client side, trusted certificates can be configured as described in Configuring Trusted Credentials

3. For more troubleshooting

- [Troubleshooting Java WS Core](#)
- [Globus Toolkit Administrator Guide - Security Section](#)

² <http://www.globus.org/cog/distribution/1.2/FAQ.TXT>

Glossary

G

grid map file A file containing entries mapping certificate subjects to local user names. This file can also serve as a access control list for GSI enabled services and is typically found in `/etc/grid-security/grid-mapfile`. For more information see the Gridmap section [here](#).

H

host credentials The combination of a host certificate and its corresponding private key.

P

private key The private part of a key pair. Depending on the type of certificate the key corresponds to it may typically be found in `$HOME/.globus/userkey.pem` (for user certificates), `/etc/grid-security/hostkey.pem` (for host certificates) or `/etc/grid-security/<service>/<service>key.pem` (for service certificates).

For more information on possible private key locations see [this](#).

T

transport-level security Uses transport-level security (TLS) mechanisms.

GT4 Java WS A&A User's Guide

DRAFT

GT4 Java WS A&A User's Guide

Introduction

Authorization. Users who run clients can programmatically set up the authorization scheme to be enforced on a per invocation basis. The properties and configuration information required depends on the configured authorization scheme.

Authentication & message-level security. Typical user configuration deals with configuring authentication mechanisms and credentials for the clients. These could be client applications, including command line clients or client configuration within services that contact other services. There are multiple mechanisms for doing this:

- Command line options (these are application-specific)
- Client security descriptors
- CoG properties
- Environment variables
- Relying on default behavior. The only default behaviors available concern the proxy file and trusted certificates locations.

More information on these mechanisms can be found in the [public interface guide](#).

Table of Contents

1. Client-side authorization	1
2. Configuring client authentication and message/transport security	2
1. Interface introduction	2
2. Syntax of the interface	4
3. Debugging	9
1. Logging in Java WS Core	9
4. Troubleshooting	11
1. Credential Troubleshooting	11
2. Error Messages For Java WS A&A	14
Glossary	17

DRAFT

List of Tables

2.1. Client side security properties	5
4.1. Credential Errors	12
4.2. Java WS A&A Errors	15

DRAFT

Chapter 1. Client-side authorization

The client side authorization can be configured for each invocation.

- Security descriptors using files. Refer to [Section 1, “Configuring Using Files”](#), specifically [Section 1.2.2, “Configuring authorization mechanism”](#).
- Security descriptors programmatically. Refer to [Section 2, “Configuring Programmatically”](#)
- Properties on the Stub. Refer to [Section 2.1, “Configuring client-side authorization on the stub”](#)

To write and configure custom authorization mechanism refer to [Section 2.2, “Writing custom client-side authorization scheme”](#).

If no authorization mechanism has been specified, HostOrSelf authorization is used. In this scheme host authorization is tried first, if it fails, self authorization is attempted

DRAFT

Chapter 2. Configuring client authentication and message/transport security

1. Interface introduction

Client-side security is set up by either setting individual properties on the `javax.xml.rpc.Stub` object used for the web service method invocation or by setting properties on a client-side security descriptor object, which in turn is propagated to client-side security handlers by making it available as a stub object property. Here are examples of the two approaches:

- Setting a property on the stub:

```
// Create endpoint reference
EndpointReferenceType endpoint = new EndpointReferenceType();
// Set address of service
String counterAddr =
    "http://localhost:8080/wsrf/services/CounterService";
// Get handle to port
CounterPortType port =
    locator.getCounterPortTypePort(endpoint);
// set client authorization to self
((Stub)port)._setProperty(Constants.AUTHORIZATION,
    SelfAuthorization.getInstance());
```

- Setting properties using a client descriptor:

```
// Client security descriptor file
String CLIENT_DESC =
    "org/globus/wsrf/samples/counter/client/client-security-config.xml";
// Create endpoint reference
EndpointReferenceType endpoint = new EndpointReferenceType();
// Set address of service
String counterAddr =
    "http://localhost:8080/wsrf/services/CounterService";
// Get handle to port
CounterPortType port =
    locator.getCounterPortTypePort(endpoint);
//Set descriptor on Stub
((Stub)port)._setProperty(Constants.CLIENT_DESCRIPTOR_FILE, CLIENT_DESC);
```

The descriptor file is described in detail in [Chapter 1, Security Descriptors Introduction](#).

**Note**

If the client needs to use transport security, the following API must be used to register the Axis transport handler for https:

```
import org.globus.axis.util.Util;
static {
    Util.registerTransport();
}
```

DRAFT

2. Syntax of the interface

DRAFT

Table 2.1. Client side security properties

Number	Task	Stub Configuration	De- Co- tion
1.	Allows for configuration of credentials for authentication.	Property: <code>org.globus.axis.gsi.GSIConstants.GSI_CREDENTIALS</code> Value equals the Instance of <code>org.ietf.jgss.GSSCredential</code> .	Sec- tion "C- ing tial
2.	Allows for configuring client-side authorization.	Property: <code>org.globus.wsrsecurity.Constants.AUTHORIZATION</code> Value equals the Instance of <code>org.globus.wsrsecurity.authorization.Authorization</code> If GSI Secure Transport or GSI Secure Conversation is used, the value should be an instance of <code>org.globus.gsi.gssapi.auth.Authorization</code> . But this translation is done automatically by the toolkit.	Rel- Sec- tion "C- ing atic ani
3.	Enable GSI Secure Conversation with specified message protection level.	1. Property: <code>org.globus.wsrsecurity.Constants.GSI_SEC_CONV</code> Values equal one of the following: <ul style="list-style-type: none"> • <code>Constants.ENCRYPTION</code> • <code>Constants.SIGNATURE</code> 2. Property: <code>org.globus.wsrsecurity.Constants.GSI_SEC_CONV_SECREPLY_UNNECESSARY</code> If the value is set to <code>Boolean.TRUE</code> , the GSI Secure conversation protection is not required in the reply message. By default, if the request was secured with GSI Secure Conversation, the response is also required to have the same protection. 3. Property: You can set the SOAP Actor of the GSI signed/encrypted SOAP message by using the <code>gssActor</code> property. We recommend that you <i>not</i> do this unless you <i>really</i> know what you are doing.	Rel- tion "C- ing cur ver

4.	Sets the GSI delegation mode. <i>Used for GSI Secure Conversation only.</i> If limited or full delegation is chosen, then some form of client-side authorization needs to be done (i.e client-side authorization cannot be set to none).	<p>Property:</p> <pre>org.globus.axis.gsi.GSIConstants.GSI_MODE</pre> <p>Value equals one of following:</p> <ol style="list-style-type: none"> 1. <code>GSIConstants.GSI_MODE_NO_DELEG</code>: No delegation is performed. 2. <code>GSIConstants.GSI_MODE_LIMITED_DELEG</code>: Limited delegation is performed. 3. <code>GSIConstants.GSI_MODE_FULL_DELEG</code>: Full delegation is performed. 	Rel tion "C ing cur ver
5.	Enables GSI Secure Transport with some protection level.	<p>Property:</p> <pre>org.globus.gsi.GSIConstants.GSI_TRANSPORT</pre> <p>Values equal one of the following:</p> <ul style="list-style-type: none"> • <code>Constants.ENCRYPTION</code> • <code>Constants.SIGNATURE</code> 	Rel tion "C ing cur por
6.	Enables anonymous authentication. <i>This option only applies to GSI Secure Conversation and GSI Transport.</i>	<p>Property:</p> <pre>org.globus.wsrp.security.Constants.GSI_ANONYMOUS</pre> <p>Value equals one of following:</p> <ol style="list-style-type: none"> 1. <code>Boolean.FALSE</code>: Anonymous authentication is disabled. 2. <code>Boolean.TRUE</code>: Anonymous authentication is enabled. 	Rel tion "C ing cur por

7.	Enable GSI Secure Message with specified message protection level.	<p>1. Property: <code>org.globus.wsrp.security.Constants.GSI_SEC_MSG</code></p> <p>Values equal one of the following:</p> <ul style="list-style-type: none"> • <code>Constants.ENCRYPTION</code> • <code>Constants.SIGNATURE</code> <p>2. Property: <code>org.globus.wsrp.security.Constants.GSI_SEC_MSG_SECREPLY_UNNECESSARY</code></p> <p>If the value is set to <code>Boolean.TRUE</code>, the GSI Secure Message protection is not required in the reply message. By default, if the request was secured with GSI Secure Message, the response is also required to have the same protection.</p> <p>3. Property: <code>org.globus.wsrp.security.Constants.GSI_SEC_MSG_SINGLECERT</code></p> <p>If the value is set to <code>Boolean.TRUE</code>, only a single certificate is used for the GSI Secure Message request. By default, the whole certificate chain is sent.</p> <p>4. Property:</p> <p>You can set the SOAP Actor of the signed message using the <code>x509Actor</code> property, but we do <i>not</i> recommend this unless you know what you are doing.</p>	Rel tion "C ing cur sag
8.	Enable WS-Security username/password authentication.	<p>Properties:</p> <p><code>org.globus.wsrp.security.Constants.USERNAME</code></p> <p>Value equals the username.</p> <p><code>org.globus.wsrp.security.Constants.PASSWORD</code></p> <p>Value equals the password.</p>	Rel tion "C ing nar wo

9.	Sets the credential that is used to encrypt the message (typically, the recipient's <i>public key</i>). <i>Used for GSI Secure Message only.</i>	<p>Property:</p> <pre>org.globus.wsrfl.impl.security.authentication .Constants.PEER_SUBJECT</pre> <p>Value equals the instance of <code>javax.security.auth.Subject</code>.</p> <p>The credential object needs to be wrapped in <code>org.globus.wsrfl.impl.security.authentication.encrypted</code> and added to the set of public credentials of the Subject object.</p> <p>For example, if <code>publicKeyFilename</code> was the file that had the recipient's public key:</p> <pre>Subject subject = new Subject(); X509Certificate serverCert = CertUtil.loadCertificate(publicKeyFilename); EncryptionCredentials encryptionCreds = new EncryptionCredentials(new X509Certificate[] { serverCert }); subject.getPublicCredentials().add(encryptionCreds); stub._setProperty(Constants.PEER_SUBJECT, subject);</pre>	Rel tion "C ing cur sag
10.	Sets the trusted certificates location.	<p>Property:</p> <pre>org.globus.wsrfl.security.TRUSTED_CERTIFICATES</pre> <p>Value should be a comma-separated list of directories and file names.</p>	Rel tion "C ing cre " -
11.	Sets the SAML Authorization Assertion to embed in SOAP Header.	<p>Property:</p> <pre>org.globus.wsrfl.impl.security.authentication.Constants.SAML_AUTHZ_ASSERTION</pre> <p>Value should be an instance of <code>org.opensaml.SAMLAssertion</code>.</p>	Car con usi des

Chapter 3. Debugging

1. Logging in Java WS Core

The following information applies to Java WS Core and all services built on Java WS Core.

Java WS Core server side has two types of loggers. One logger is used for development logging and by default writes to standard out. The other logger includes system administration information and is [CEDPs best practices](#)¹ compliant.

On client side, only developer logging is available and is configured using `log4j.properties`.

1.1. Development Logging in Java WS Core

The following information applies to Java WS Core and those services built on it.

Logging in the Java WS Core is based on the [Jakarta Commons Logging](#)² API. Commons Logging provides a consistent interface for instrumenting source code while at the same time allowing the user to plug-in a different logging implementation. Currently we use [Log4j](#)³ as a logging implementation. Log4j uses a separate configuration file to configure itself. Please see Log4j documentation for details on the [configuration file format](#)⁴.

1.1.1. Configuring server side developer logs

Server side logging can be configured in `$GLOBUS_LOCATION/container-log4j.properties`, when the container is stand alone container. For tomcat level logging, refer to [Logging for Tomcat](#)⁵. The logger `log4j.appender.A1` is used for developer logging and by default writes output to the system output. By default it is set for all warnings in the Globus Toolkit package to be displayed.

Additional logging can be enabled for a package by adding a new line to the configuration file. Example:

```
#for debug level logging from org.globus.package.FooClass
log4j.category.org.globus.package.name.FooClass=DEBUG
#for warnings from org.some.warn.package
log4j.category.org.some.warn.package=WARN
```

1.1.2. Configuring client side developer logs

Client side logging can be configured in `$GLOBUS_LOCATION/log4j.properties`. The logger `log4j.appender.A1` is used for developer logging and by default writes output to the system output. By default it is set for all warnings in the Globus Toolkit package to be displayed.

¹ <http://cedps.net/index.php/LoggingBestPractices>

² <http://jakarta.apache.org/commons/logging/>

³ <http://logging.apache.org/log4j/>

⁴ [http://logging.apache.org/log4j/docs/api/org/apache/log4j/PropertyConfigurator.html#doConfigure\(java.lang.String,org.apache.log4j.spi.LoggerRepository\)](http://logging.apache.org/log4j/docs/api/org/apache/log4j/PropertyConfigurator.html#doConfigure(java.lang.String,org.apache.log4j.spi.LoggerRepository))

⁵ <http://tomcat.apache.org/tomcat-5.5-doc/logging.html>

1.2. Configuring system administration logs

The specific logger to edit will be `log4j.logger.sysadmin` in `$GLOBUS_LOCATION/container-log4j.properties`. There you can configure the following properties:

```
log4j.appender.infoCategory=org.apache.log4j.RollingFileAppender
log4j.appender.infoCategory.Threshold=INFO
log4j.appender.infoCategory.File=var/containerLog
log4j.appender.infoCategory.MaxFileSize=10MB
log4j.appender.infoCategory.MaxBackupIndex=2
```

Above implies the logging file is rolling with each file size limited to 10MB and the logging information is stored in `$GLOBUS_LOCATION/var/containerLog`.

1.3. Sample log file

The [sample log file](#)⁶ contains many log entries for various scenarios in the Java WS container.

⁶ <http://www.globus.org/toolkit/docs/4.2/4.2.0/common/javawscore/sample-container-log.txt>

Chapter 4. Troubleshooting

For a list of common errors in GT, see [Error Codes](#).

For information about system administrator logs, see [Chapter 7, Troubleshooting](#).

1. Credential Troubleshooting

1.1. Credential Errors

The following are some common problems that may cause clients or servers to report that credentials are invalid:

For a list of common errors in GT, see [Error Codes](#).

DRAFT

Table 4.1. Credential Errors

Error Code	Definition	Possible Solutions
Your proxy credential may have expired	Your proxy credential may have expired.	Use <code>grid-proxy-info</code> to check whether the proxy credential has actually expired. If it has, generate a new proxy with <code>grid-proxy-init</code> .
The system clock on either the local or remote system is wrong.	This may cause the server or client to conclude that a credential has expired.	Check the system clocks on the local and remote system.
Your end-user certificate may have expired	Your end-user certificate may have expired	Use <code>grid-cert-info</code> to check your certificate's expiration date. If it has expired, follow your CA's procedures to get a new one.
The permissions may be wrong on your proxy file	If the permissions on your proxy file are too lax (for example, if others can read your proxy file), Globus Toolkit clients will not use that file to authenticate.	You can "fix" this problem by changing the permissions on the file or by destroying it (with <code>grid-proxy-destroy</code>) and creating a new one (with <code>grid-proxy-init</code>). Important: However, it is still possible that someone else has made a copy of that file during the time that the permissions were wrong. In that case, they will be able to impersonate you until the proxy file expires or your permissions or end-user certificate are revoked, whichever happens first.
The permissions may be wrong on your private key file	If the permissions on your end user certificate private key file are too lax (for example, if others can read the file), <code>grid-proxy-init</code> will refuse to create a proxy certificate.	You can "fix" this by changing the permissions on the private key file. Important: However, you will still have a much more serious problem: it is possible that someone has made a copy of your private key file. Although this file is encrypted, it is possible that someone will be able to decrypt the private key, at which point they will be able to impersonate you as long as your end user certificate is valid. You should contact your CA to have your end-user certificate revoked and get a new one.
The remote system may not trust your CA	The remote system may not trust your CA	Verify that the remote system is configured to trust the CA that issued your end-entity certificate. See Installing GT 4.2.0 for details.
You may not trust the remote system's CA	You may not trust the remote system's CA	Verify that your system is configured to trust the remote CA (or that your environment is set up to trust the remote CA). See Installing GT 4.2.0 for details.
There may be something wrong with the remote service's credentials	There may be something wrong with the remote service's credentials	It is sometimes difficult to distinguish between errors reported by the remote service regarding your credentials and errors reported by the client interface regarding the remote service's credentials. If you cannot find anything wrong with your credentials, check for the same conditions on the remote system (or ask a remote administrator to do so).

1.2. Some tools to validate certificate setup

1.2.1. Check that the user certificate is valid

```
openssl verify -CApath /etc/grid-security/certificates
-purpose sslclient ~/.globus/usercert.pem
```

1.2.2. Connect to the server using s_client

```
openssl s_client -ssl3 -cert ~/.globus/usercert.pem -key
~/.globus/userkey.pem -CApath /etc/grid-security/certificates
-connect <host:port>
```

Here `<host:port>` denotes the server and port you connect to.

If it prints an error and puts you back at the command prompt, then it typically means that the *server* has closed the connection, i.e. that the server was not happy with the client's certificate and verification. Check the SSL log on the server.

If the command "hangs" then it has actually opened a telnet style (but secure) socket, and you can "talk" to the server.

You should be able to scroll up and see the subject names of the server's verification chain:

```
depth=2 /DC=net/DC=ES/O=ESnet/OU=Certificate Authorities/CN=ESnet Root CA 1
verify return:1
depth=1 /DC=org/DC=DOEGrids/OU=Certificate Authorities/CN=DOEGrids CA 1
verify return:1
depth=0 /DC=org/DC=doegrids/OU=Services/CN=wiggum.mcs.anl.gov
verify return:1
```

In this case, there were no errors. Errors would give you an extra line next to the subject name of the certificate that caused the error.

1.2.3. Check that the server certificate is valid

Requires root login on server:

```
openssl verify -CApath /etc/grid-security/certificates -purpose sslserver
/etc/grid-security/hostcert.pem
```

2. Error Messages For Java WS A&A

DRAFT

Table 4.2. Java WS A&A Errors

Error Code	Definition	Possible Solutions
[JWSSEC-248] Secure container requires valid credentials	This error occurs when <code>globus-start-container</code> is run without any valid credentials. Either a proxy certificate or service/host certificate needs to be configured for the container to start up.	<ol style="list-style-type: none"> <li data-bbox="805 300 1334 604">1. If you are not looking to start up a container that uses GSI Secure Transport, which is used by the container by default, use <code>globus-start-container -nosec</code>. You will be able to use insecure clients and services. However, this also implies that if you have not configured individual services with credentials, you will not be able to securely access the service. <li data-bbox="805 615 1334 804">2. If you are running a personal container, generate a proxy certificate with <code>grid-proxy-init</code>. If the proxy certificate is not in the default location, configure the container security descriptor as described in Configuring Container Security Descriptor. <li data-bbox="805 835 1334 930">3. If you want to use host certificates, configure the container security descriptor as described Configuring Credentials.
Failed to start container: Container failed to initialize [Caused by: [JWSSEC-250] Failed to load certificate/key file]	This error occurs if the file path to the container certificate and key configured are invalid.	<ol style="list-style-type: none"> <li data-bbox="805 963 1334 1163">1. The path to the container certificate and key are configured in <code>\$GLOBUS_LOCATION/etc/globus_wsrf_core/global_security_descriptor.xml</code>. This file is loaded as described [here - fixme link]. Ensure that the path is correct.
Failed to start container: Container failed to initialize [Caused by: [JWSSEC-249] Failed to load proxy file]	This error occurs if container proxy file configured is invalid.	<ol style="list-style-type: none"> <li data-bbox="805 1194 1334 1394">1. The path to the container proxy certificates are configured in <code>\$GLOBUS_LOCATION/etc/globus_wsrf_core/global_security_descriptor.xml</code>. This file is loaded as described [here - fixme link]. Ensure that the path is correct.
Failed to start container: Container failed to initialize [Caused by: [JWSSEC-245] Error parsing file: "etc/globus_wsrf_core/global_security_descriptor.xml" [Caused by: ...]	This error occurs if the container security descriptor configured is invalid.	<ol style="list-style-type: none"> <li data-bbox="805 1425 1334 1520">1. The container security descriptor should conform to the Container Security Descriptor Schema.¹ <li data-bbox="805 1551 1334 1604">2. Refer to the "Caused by: " section for details on the specific element that is not correct.

¹ http://www.globus.org/toolkit/docs/4.2.0/security/container_security_descriptor.xsd

Error Code	Definition	Possible Solutions
[JGLOBUS-77] Unknown CA	This error occurs if the CA certificate for the credentials being used is not installed correctly.	<ol style="list-style-type: none"><li data-bbox="805 243 1336 401">1. If this issue occurs on the server side, the container is not configured with CA certificates. The container looks for trusted certificates in the default location as described Java CoG Toolkit FAQ²<li data-bbox="805 428 1336 520">2. On the server side, the trusted certificates can be configured as described in Trusted Certificates<li data-bbox="805 548 1336 640">3. On the client side, trusted certificates can be configured as described in Configuring Trusted Credentials

² <http://www.globus.org/cog/distribution/1.2/FAQ.TXT>

Glossary

some terms not in the docs but wanted in glossary: scheduler

P

public key The public part of a key pair used for cryptographic operations (e.g. signing, encrypting).

S

scheduler Term used to describe a job scheduler mechanism to which GRAM interfaces. It is a networked system for submitting, controlling, and monitoring the workload of batch jobs in one or more computers. The jobs or tasks are scheduled for execution at a time chosen by the subsystem according to an available policy and availability of resources. Popular job schedulers include Portable Batch System (PBS), Platform LSF, and IBM LoadLeveler.

DRAFT

GT 4.2.0 Java WS A&A Developer's Guide

DRAFT

GT 4.2.0 Java WS A&A Developer's Guide

Introduction

fixme

DRAFT

Table of Contents

1. Overview	1
1. Authentication overview	1
2. Authorization framework overview	1
2. Before you begin	2
1. Feature Summary	2
2. Tested platforms	3
3. Backward compatibility summary	3
4. Technology dependencies	3
5. Security considerations for Java WS A&A	4
3. Usage scenarios	6
1. Delegation	6
2. Embedding Key Information in EPRs	6
3. Obtaining peer credentials on the server side	6
4. Obtaining peer credentials from message context on the client side	7
5. Using SAML Authorization Assertions	8
6. Using Multiple Message Protection Schemes	9
4. Tutorials	10
5. Architecture and design overview	11
1. Authentication/message-level architecture	11
2. Authorization architecture	15
6. APIs	17
1. Authorization Programming Model	17
2. Authentication/message protection Programming Model	17
3. API	17
7. Services and WSDL	19
1. Secure Conversation Service	19
2. SAML Authorization Callout	21
8. Framework-level Protocols	23
1. WS-Security	23
2. Transport (HTTPS) Security	23
9. Configuring client authentication and message/transport security	24
1. Interface introduction	24
2. Syntax of the interface	26
10. Authorization domain-level interface	31
1. Interface introduction	31
2. Syntax of the interface	31
11. Configuring	36
1. Configuring authorization	36
2. Configuring authentication/message protection	37
12. Environment variable interface	38
1. Environmental variables for WS Authentication & Authorization (Java)	38
13. PDP Reference	39
1. Introduction	39
2. Access Control List PDP	39
3. GridMapAuthorization	40
4. Host Authorization	41
5. IdentityAuthorization	42
6. No Authorization	42
7. ResourceProperties Authorization	43
8. SAML Authorization Callout	44
9. SAML Authorization Assertion PDP	46

10. Self Authorization	46
11. Username Authorization	47
14. PIP Reference	51
1. Introduction	51
2. Container PIP	51
3. X509Bootstrap	53
4. SAML Authorization Assertion PIP	54
5. Parameter PIP	55
15. Debugging	57
1. Debugging authorization	57
16. Troubleshooting	59
1. Error Messages For Java WS A&A	60
17. Related Documentation	63
Glossary	64

DRAFT

List of Figures

5.1. The new certificate is signed by the owner, rather than a CA. 12
5.2. JAX-RPC handlers involved in security related message processing on a server. 14

DRAFT

List of Tables

9.1. Client side security properties	27
11.1. Configuring server side authentication and message/transport security	37
14.1. Attribute I	52
14.2. Attribute II	52
14.3. Attribute III	52
14.4. Attribute I	52
14.5. Attribute II	53
14.6. Attribute III	53
14.7. Attribute I	54
14.8. Attribute II	54
14.9. Attribute I	55
14.10. Attribute II	55
14.11. Attribute I	56
16.1. Java WS A&A Errors	61

DRAFT

Chapter 1. Overview

1. Authentication overview

Java WS authentication contains mainly framework-level code and, as such, developing services and clients utilizing this component does in general involve either programmatically or declaratively driving the framework-level security code.

Now, what does this entail? On the programmatic side of things, it involves acquiring credentials, passing these credentials on to the framework, and setting various authentication- and protection-related flags, either in a descriptor or as properties on a stub object. On the declarative side, it involves setting up security descriptors, both client and service side, to prescribe the security policy used to drive the security framework code.

2. Authorization framework overview

The authorization framework enforces the configured authorization policy on the service and client side.

On the service side, the framework allows developers to configure a chain of authorization mechanisms either programmatically or declaratively using security descriptors. It also allows for plugging in new authorization schemes (in addition to using those that are provided with the framework). Moreover, the framework allows for this configuration to be done at resource, service or container level, each taking precedence in the order specified and scoped as the name suggests.

On the client side, a pluggable framework for authorization of service is provided.

Chapter 2. Before you begin

1. Feature Summary

1.1. Authentication/message protection features

Features new in GT 4.2.0

None.

Other Supported Features

- Compliance with published IBM/Microsoft WS-Trust and WS-SecureConversation specifications
- Compliance with the Web Services Security 1.0 standard
- HTTPS support
- Message encryption, integrity protection and replay attack prevention
- Establishment of a session key for light-weight message protection

Deprecated Features

- None.

1.2. Authorization features

Features new in GT 4.2.0:

- *Enhanced server-side attributed-based authorization framework:* The server-side authorization framework has been reworked to support attribute based authorization with delegation of rights. The framework allows for configuring a chain of Policy Information Points(PIPs) and Policy Decision Points(PDPs) and a combining algorithm that processes the individual decisions returned by the PDPs. Some of the key changes from the previous versions are:
 - Java Server side authorization framework has been moved to an independent module. Refer to Changes Summary for details.
 - Authorization framework uses a set of attributes to identify entities
 - The authorization engine uses Java Security provider framework to allow different combining algorithms to be plugged in.
 - A default implementation of permit override combining algorithm, which looks for a permit decision chain, to allow for fine grained delegation of rights.

Refer [Chapter 5, Architecture and design overview](#) for detailed information on the architecture.

- *Host or Self Authoriation:* Support for a pluggable PDP that does host authorization, and if that fails, tries self authorization.

- The security descriptor framework, used to configure security properties for the security framework has been enhanced. Detailed information about the framework is provided [Java WS A&A Security Descriptor Framework](#).

Other Supported Features

- Authorization based on `grid-mapfile` and other access control lists.
- Ability to implement custom authorization modules.
- A SAML callout authorization module enables outsourcing of authorization decisions to an authorization service (e.g. PERMIS).

Deprecated Features

- None

2. Tested platforms

Java WS A&A should work on any platform that supports J2SE 1.3.1 or higher.

Tested Platforms for Java WS A&A:

- Linux (Red Hat 7.3)
- Windows 2000
- Solaris 9

3. Backward compatibility summary

3.1. Authentication compatibility

Since GT 4.0.x release, some incompatible changes have been made:

- Security Descriptors: The security descriptor schema has changed since GT 4.0.x and the descriptors from GT 4.0.x cannot be used as is.
- Secure Conversation port type: The WS Addressing version in Java WS Core has been updated and the secure conversation port type has changed to reflect this. Therefore, GT 4.0.x secure conversation clients are incompatible with GT 4.2.x servers and vice versa.

3.2. Authorization compatibility

The authorization framework has been reworked as described in [Change Summary](#). The configuration and authorization interfaces have since changed and a [Migration Guide](#) is provided.

4. Technology dependencies

Java WS A&A depends on the following GT components:

- Java WS Core.

Authentication and message-protection depends on the following 3rd party software:

- Apache WSFX Security Libraries
- PureTLS Libraries
- BouncyCastle JCE provider
- Cryptix Libraries
- Apache XML Security Libraries

The authorization framework depends on the following 3rd party software:

- OpenSAML

5. Security considerations for Java WS A&A

5.1. Security considerations for authorization

5.1.1. Client side authorization

Client authorization of the server is done after the completion of the operation when GSI Secure Message authentication is used. If you require client authorization to be done prior, use GSI Secure Conversation or GSI *Transport security*.

5.1.2. Host authorization

During host authorization, the toolkit treats DNS "hostname-*.edu" as equivalent to "hostname.edu". This means that if a service was setup to do host authorization and hence accept the certificate "hostname.edu", it would also accept certificates with DNS "hostname-*.edu".

The feature is in place to allow a multi-homed host following a "hostname-interface" naming convention, to have a single host certificate. For example, host "grid.test.edu" would also accept likes of "grid-1.test.edu" or "grid-foo.test.edu".



Note

The wildcard character "*" matches only name of the host and not domain components. This means that "hostname.edu" will not match "hostname-foo.sub.edu", but will match "host-foo.edu".



Note

If a host was set up to accept "hostname-1.edu", it will not accept any of "hostname-*.edu".

A [bug](#)¹ has been opened to see if this feature needs to be modified.

5.2. Security considerations for Message/Transport-level Security

5.2.1. File permissions

The Java security code currently does not enforce secure permissions and, implicitly, file ownership requirements on any of the security related files, e.g. configuration and credential files. It is thus important that administrators ensure

¹ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2969

that the relevant files have correct permissions and ownership. Permissions should generally be as restrictive as possible, i.e. *private keys* should be readable only by the file owner and other files should be writable by owner only, and the files should generally be owned by the globus user (the requirements that the C code enforces are documented in [Configuring GSI](#)).

Also refer to [Section 5, “Known Problems”](#) for details on any other open issues.

DRAFT

Chapter 3. Usage scenarios

1. Delegation

There are two ways a client can delegate its credential to a service:

- using Delegation Service, and
- using GSI Secure Conversation.

A client can delegate using the [Delegation Service](#). This method is independent of the security scheme used and can be reused across multiple invocations of the client to multiple services (provided the services are in the same hosting environment as the Delegation Service). The link provided has details on client-side steps to delegate and service-side code to get the delegated credential.

GSI Secure Conversation has delegation built into the protocol. Delegation can be requested by setting the `GSIConstants.GSI_MODE` property on the Stub or using security descriptors as described in [Section 1.2.3, “Configuring GSI Secure Conversation”](#). If full or limited delegation is performed, the client credential can be obtained from the message context as follows:

```
Subject subject = (Subject) msgCtx.getProperty(Constants.PEER_SUBJECT);
```

The server can be configured such that container, service or client credentials are used for the operation invoked. For the client credentials to be used, the client should have delegated the credentials. Configuring this option is described in [Section 1.6.2, “Configuring the run-as mode”](#). Note that this is a server-side configuration. If `caller-identity` is chosen for the `run-as` configuration and the client's credentials have been successfully delegated, then the delegated credentials are associated with the current thread. The credentials in this case can be obtained as follows:

```
Subject subject = JaasSubject.getCurrentSubject();
```

2. Embedding Key Information in EPRs

GT provides an API to embed key information in an Endpoint Reference, as defined in the OGSA Basic Security Profile. The key information is embedded in the extensibility element of the EPR rather than the meta-data element as defined in the specification, since the toolkit uses older version of the WS Addressing specification.

This information would be useful to ascertain the expected identity of the service for authorizing the service or to get the public certificate of the resource to be used for encrypting the request to the service. The optional `usage` element in the embedded key information indicates the use of the embedded keys, either for signature or encryption.

The API is in class `org.globus.wsrfl.impl.security.util.EPRUtil`. The method to embed the certificates is called `insertCertificates` and the method to extract the key information is called `extractCertificates`. Please refer to [API documentation](#) for details on using the methods.

3. Obtaining peer credentials on the server side

The security handlers populate a Subject object with peer information. The following code can be used to access the peer credentials. Note that the message context needs to be associated with the thread.

```
import org.globus.wsrfl.security.SecurityManager;
```

```
import javax.security.auth.Subject;

org.apache.axis.MessageContext mctx =
org.apache.axis.MessageContext.getCurrentContext();
SecurityManager manager = SecurityManager.getManager(mctx);
Subject subject = manager.getPeerSubject();
```

The following code snippet shows how the certificate chain can be extracted from the peer subject:

```
java.util.Set set =
subject.getPublicCredentials(X509Certificate[].class);
Iterator iterator = set.iterator();
while (iterator.hasNext()) {
X509Certificate[] certArray = (X509Certificate[]) iterator.next();
System.out.println("Cert array " + certArray.length);
}
```

To obtain the peer principal (for example, the Distinguished Name from X509 Certificate), the following code snippet can be used:

```
org.apache.axis.MessageContext mctx =
org.apache.axis.MessageContext.getCurrentContext();
SecurityManager manager = SecurityManager.getManager(mctx);
Principal principal = manager.getCallerPrincipal();
String caller = manager.getCaller();
```

If credentials were delegated as described above, private credentials are also populated:

```
java.util.Set set = subject.getPrivateCredentials();
```

4. Obtaining peer credentials from message context on the client side

- **GSI Secure Conversation:** With this mechanism, the peer credentials can be obtained once the handshake is completed:

```
import org.globus.wsrp.impl.security.authentication.Constants;
import org.globus.wsrp.impl.security.authentication.secureconv.service.S
import org.ietf.jgss.GSSContext;
import org.globus.gsi.gssapi.GSSContextants;

// Get current secure context from message context
SecurityContext secContext =
messageContext.getProperty(Constants.CONTEXT);
GSSContext gssContext = secContext.getContext();
```

```
Vector peerCerts =  
gssContext.inquireByOid(GSSContants.X509_CERT_CHAIN);
```

- **GSI Secure Transport:** With this mechanism, the peer credentials can be obtained once the handshake is completed:

```
import org.ietf.jgss.GSSContext;  
import org.globus.gsi.gssapi.GSSContants;  
import org.globus.wsrfl.impl.security.authentication.Constants;  
  
// Get current secure context from message context  
GSSContext gssContext =  
messageContext.getProperty(Constants.TRANSPORT_SECURITY_CONTEXT);  
Vector peerCerts =  
gssContext.inquireByOid(GSSContants.X509_CERT_CHAIN);
```

- **GSI Secure Message:** With this mechanism, the peer credentials can be obtained only when the response is received:

```
import org.globus.wsrfl.impl.security.authentication.Constants;  
  
// Get peer subject from current message context  
Subject subject =  
(Subject) messageCtx.getProperty(Constants.PEER_SUBJECT);  
Set peerCerts =  
subject.getPublicCredentials(X509Certificate[].class);
```

5. Using SAML Authorization Assertions

SAML Authorization assertions can be used to transport authorization decision statements to the point of enforcement. Such assertions can be obtained from any entity that manages use rights. For example, [Community Authorization Service \(CAS\)](#), a Globus Toolkit higher level service, can be used to manage user rights and issue such assertions. On the remote side, PDPs/PIPs might be used to extract the assertion and enforce the rights. Some PDPs/PIPs with such functionality are distributed and details can be found in [Chapter 14, PIP Reference](#) and [Chapter 13, PDP Reference](#).

The toolkit provides two mechanisms to do this:

- **Using proxy certificates:** In this mechanism the SAML Authorization Assertions are embedded as non-critical assertions in the proxy certificate. These assertions can then be extracted from the certificate, to enforce access rights at the resource.

A command line client `$GLOBUS_LOCATION/bin/globus-embed-assertion` implemented using `org.globus.wsrfl.client.EmbedAssertion` can be used to embed a SAML Assertion stored in a file into a credential.

- **Using SOAP Header:** The SAML Assertion can also be embedded in the SOAP Header. The toolkit uses the WS-Security SAML Token Profile to embed the assertion.

Section 2, “[Syntax of the interface](#)” describes the configuration required to embed an assertion into the message header. At the remote end, the security handlers detect the presence of the SAML Assertion in the header and store it as a string in the Message Context property `Constants.SAML_ASSERTION_STR`.

6. Using Multiple Message Protection Schemes

Multiple message protection schemes can be used in a single invocation, although it is worth noting that this will cause a performance penalty.

For example, both Secure Transport and Secure Conversation can be done on the same invocation by using the following:

```
stub._setProperty(Constants.GSI_SEC_CONV, Constants.INTEGRITY);
stub._setProperty(Constants.GSI_TRANSPORT, Constants.PRIVACY);
```



Note

These two mechanisms share a single property for authorization. There is a bug open to provide independent support: [Bug 4350](#)¹

Similarly Secure Messages can be used in tandem with other message protection mechanisms.

¹ http://bugzilla.mcs.anl.gov/globus/show_bug.cgi?id=4350

Chapter 4. Tutorials

There are no tutorials available at this time.

DRAFT

Chapter 5. Architecture and design overview

1. Authentication/message-level architecture

1.1. Transport Security

The toolkit by default is deployed with our implementation of transport security, which is based on HTTP over SSL, also known as HTTPS, with modifications to path validation to enable X.509 *Proxy Certificate* support. In contrast to the GT3 version of the toolkit, the default transport security enabled in the toolkit does not support delegation of proxy certificates as part of the security handshake.

However, the underlying security libraries and handlers required for secure transport with delegation, also known as HTTPG, is still supported and shipped as part of the CoG library. The GT4 Java WS code base and configuration can be modified to use the HTTPG protocol as required.

Transport security is implemented by layering on top of the *GSISocket* class provided in JGlobus. This class deals with the security-related aspects of connection establishment as well as message protection. The socket interface serves as an abstraction layer that allows the HTTP protocol handling code to be unaware of the underlying security properties of the connection.

Container-level credentials are required and, irrespective of security settings on the service being accessed, these credentials are used for the handshake.

1.1.1. Server-Side Security

On the server-side, transport security is enabled by simply switching a non-secure socket implementation with the *GSISocket* implementation. In addition to this change, some code was added to propagate authentication information and message protection settings to the relevant security handlers, in particular the authorization and security policy handlers.

1.1.2. Client-Side Security

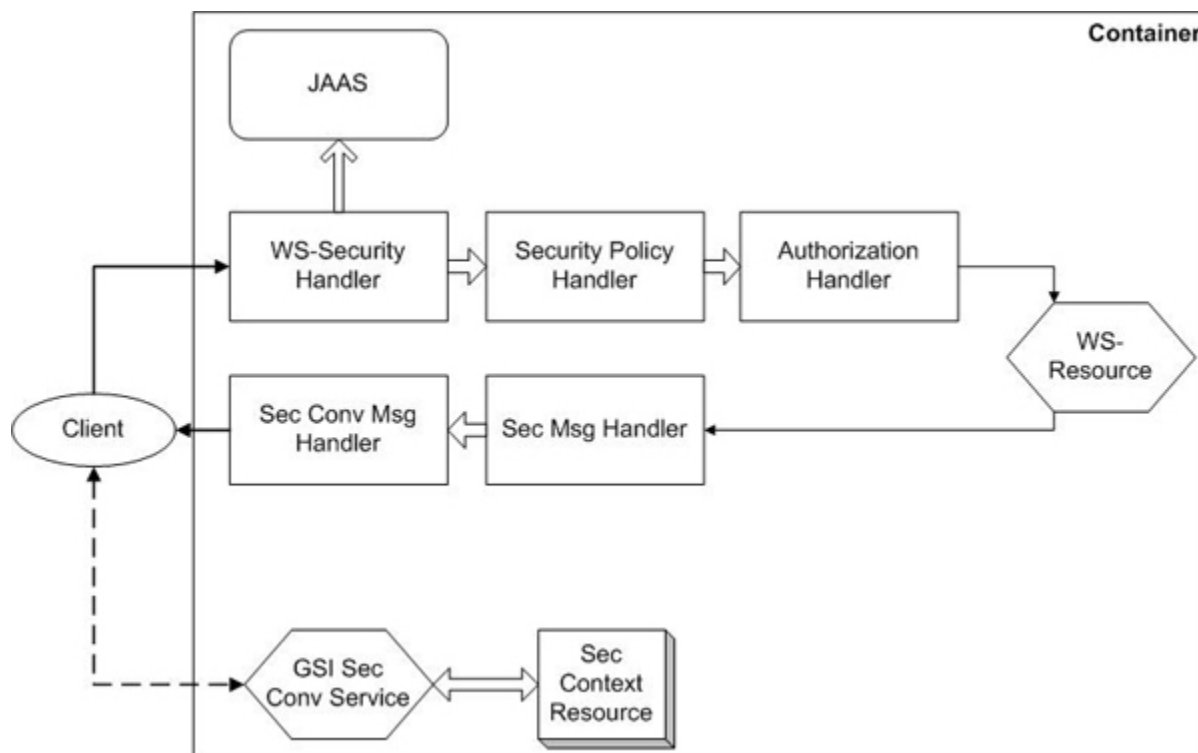
On the client-side, transport security is similarly enabled by switching a non-secure socket implementation with the *GSISocket* implementation and registering a protocol handler for HTTPS that uses the secure socket implementation. In practice, this means that any messages targeted at an HTTPS endpoint will, irregardless of any stub properties, be authenticated and protected. It also means that any messages sent to an HTTP endpoint will not be secured, again irregardless of any stub properties. Stub properties are only used to communicate the desired message protection level, i.e. either integrity only or integrity and privacy.

1.2. Message Level Security

1.2.1. Server Side Security

This section aims to describe the message flow and processing that occurs for a security-enabled service. The figure below shows the JAX-RPC handlers that are involved in security-related message processing on a server.

Figure 5.1. The new certificate is signed by the owner, rather than a CA.



GT4 provides two mechanisms, **GSI Secure Conversation** and **GSI Secure Message** security, for authentication and secure communication.

- In the GSI Secure Conversation approach the client establishes a context with the server before sending any data. This context serves to authenticate the client identity to the server and to establish a shared secret using a collocated GSI Secure Conversation Service. Once the context establishment is complete, the client can securely invoke an operation on the service by signing or encrypting outgoing messages using the shared secret captured in the context.
- The GSI Secure Message approach differs in that no context is established before invoking an operation. The client simply uses existing keying material, such as an X509 *End Entity Certificate*, to secure messages and authenticate itself to the service.

Securing of messages in the GSI Secure Conversation approach, i.e. using a shared secret, requires less computational effort than using existing keying material in the GSI Secure Message approach. This allows the client to trade off the extra step of establishing a context to enable more computationally efficient messages protection once that context has been established.

1.2.2. Message Processing

When a message arrives from the client, the SOAP engine invokes several security-related handlers:

1. The first of these handlers, the **WS-Security handler**, searches the message for any WS-Security headers. From these headers, it extracts any keying material, which can be either in the form of an X509 certificate and associated certificate chain or a reference to a previously established secure conversation session. It also checks any signatures and/or decrypts elements in the SOAP body. The handler then populates a peer JAAS subject object with principals and any associated keying material whose veracity was ascertained during the signature checking or decryption step.

2. The next handler that gets invoked, the **security policy handler**, checks that incoming messages fulfill any security requirements the service may have. These requirements are specified, on a per-operation basis, as part of a [security descriptor](#) during service deployment. The security policy handler will also identify the correct JAAS subject to associate with the current thread of execution. Generally, this means choosing between the peer subject populated by the WS-Security handler, the subject associated with the hosting environment and the subject associated with the service itself. The actual association is done by the pivot handler, a non-security handler not shown in the figure that handles the details of delivering the message to the service.
3. The security policy handler is followed by an **authorization handler**. This handler verifies that the principal established by the WS-Security handler is authorized to invoke the service. The type of authorization that is performed is specified as part of a deployment descriptor. More information can be found in the [authorization framework documentation](#).

Once the message has passed the authorization handler, it is finally handed off to the actual service for processing (discounting any non-security-related handlers, which are outside the scope of this document).

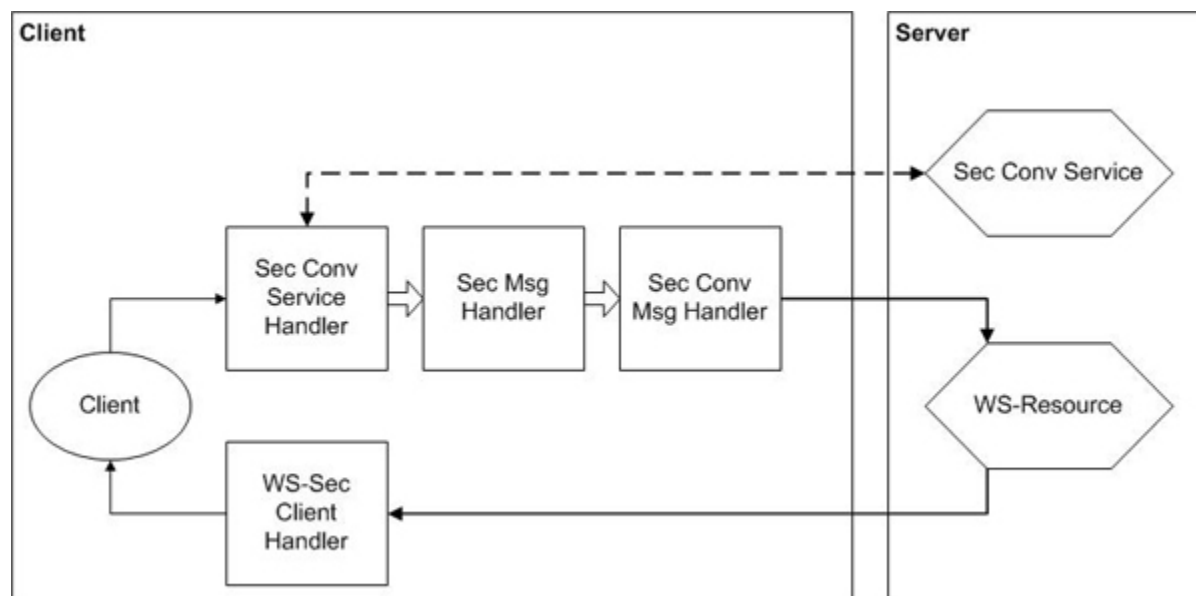
Replies from the service back to the client are processed by two outbound handlers: the GSI Secure Conversation message handler and the GSI Secure Message handler. The GSI Secure Conversation message handler deals with encrypting and signing messages using a previously established security context, whereas the GSI Secure Message handler deals with messages by signing or encrypting the messages using X509 certificates.

The operations that are actually performed depend on the message properties associated with the message by the inbound handlers, i.e. outbound messages will have the same security attributes as inbound messages. That being said, a service has the option of modifying the message properties, if so desired. These handlers are identical to the client-side handlers described in the following section.

1.2.3. Client Side Security

This section describes the security-related message processing for Java-based GT4 clients. In contrast to the server side, where security is specified via deployment descriptors, client side security configuration is handled by the application. This means that a client-side application must explicitly pass information to the client-side handlers on what type of security to use. This is also true for the case of services acting as clients. The below figure shows the JAX-RPC handlers that are involved in security-related message processing on a server.

Figure 5.2. JAX-RPC handlers involved in security related message processing on a server.



1.2.4. Message Processing

The client-side application can specify the use of either the GSI Secure Conversation security approach or the GSI Secure Message security approach. It does this by setting a per-message property that is processed by the client-side security handlers.

There are three outbound client-side security handlers:

1. The **secure conversation service handler** is only operational if GSI Secure Conversation mode is in use. It establishes a security session with a secure conversation service collocated with the service with which the client aims to communicate. When the client sends the initial message to the service with a property indicating that session-based security is required, this handler intercepts the message and establishes a security session. It will also authorize the service by comparing the service's principal/subject obtained during session establishment with a value provided by the client application. Once the session has been established, the handler passes on the original message for further processing.
2. The next handler in the chain, the **secure message handler**, is only operational if GSI Secure Message mode is in use. It signs and/or encrypts messages using X.509 credentials.
3. The third outbound handler [fixme - is there a name?] is operational only if GSI Secure Conversation mode is in use. It handles signing and/or encryption of messages using a security session established by the first handler.

The client-side inbound handler (the WS-Security client handler) deals with verifying and decrypting any signed and/or encrypted incoming messages. In the case of the GSI Secure Message operation, it will also authorize the remote side in a similar fashion to the outbound secure conversation service handler.

2. Authorization architecture

2.1. Server-side authorization

The Java WS Authorization framework leverages the generic GT Java Authorization Framework. The framework consists of an engine that evaluates a chain of configured authorization schemes, also known as Policy Decision Points (PDPs), to determine if the client making the invocation can access the resource/service. The chain can also be configured with Policy Information Points (PIPs), which can be used to glean information that would be useful in the decision making process.

The framework enables attribute-based authorization. PIPs can be used to collect attributes about resource/operations/subjects and used in the decision making process. While the toolkit provides some implementations of PIPs/PDPs, the framework is pluggable and custom mechanisms can be written and configured.

An authorization engine consists of PIPs, PDPs and a combining algorithm. The configured authorization engine is invoked as part of a handler chain, immediately after authentication of the invocation (`java:org.globus.ws-rf.impl.security.authorization.AuthorizationHandler`). If no security mechanism is used for an operation, authorization is not done for that operation.

The architecture of Generic Java Authorization Engine is described in detail in this [document](#)¹. It also describes interfaces and writing custom PDPs/PIPs.

Any PDP has to implement the interface `org.globus.security.authorization.PDP` and contain the logic to return a permit or deny based on information such as the subject DN, the service being accessed and the operation being performed. To manage configuration information, each PDP can be bootstrapped with an object implementing the `org.globus.security.authorization.ChainConfig` interface. The interface has get and set methods which can be used to retrieve and set scoped parameters for the PDP.

PIPs have to implement the interface `org.globus.security.authorization.PIP` with the functionality to collect attributes from the invocation context that are relevant to making the authorization decision.

2.1.1. Authorization Policy Configuration

A chain of PDPs and PIPs, with relevant configuration information, can be configured at resource, service or container level. These chain use Permit Override with Delegation as default combining algorithm. Additionally an administrative policy can be configured at the container level. The administrative chain uses First Applicable combining algorithm by default. Note that cominging algorithms can be configured to over-ride the deafult. The following describes the precedence in which configured policy is used:

1. If container level administrative policy is specified, it is evaluated.
 - a. If (1) returns a deny, the request is denied.
 - b. If (1) returns a permit, step (2) is done.
2. If resource level policy is specified, it is evaluated.
 - a. If (2) returns a deny, the request is denied.
 - b. If (2) returns a permit, the request is permitted.
3. If (2) is not specified and service level policy is specified, it is evaluated.

¹ ../gtJavaAuthzEngine.pdf

- a. If (3) returns a deny, the request is denied.
 - b. If (3) returns a permit, the request is permitted.
4. If (3) is not specified and container level policy configuration is specified, it is evaluated.
 - a. If (4) returns a deny, the request is denied.
 - b. If (4) returns a permit, the request is permitted.

2.1.2. Authorization Handler Steps

1. Invoke Container PIP to collect attributes inherent to the framework. The PIP creates an instance of RequestEntities class to use as parameter with PIPs. It also creates an instance of ResourceChainConfig class to push the current message context as a parameter to ContainerPIP.
2. Evaluate the administrator authorization engine, if one is configured
 - a. If bootstrap overwrite is configured, then only BootstrapPIPs in administrator engine is invoked. Else the X509 Bootstrap PIP is invoked prior to any other Bootstrap PIPs configured.
 - b. The authorization engine is run and if a deny decision is returned, the operation is denied. If a permit decision is returned, the operation is permitted. If a not applicable or indeterminate is returned, further authorization engines are evaluated.
3. Evaluate the authorization engine configured in the resource, service, container, in that order depending on which is configured
 - a. If bootstrap overwrite is configured, then only BootstrapPIPs in administrator engine is invoked. Else the X509 Bootstrap PIP is invoked prior to any other Bootstrap PIPs configured.
 - b. If any decision other than a Permit is returned, the operation is denied. If a permit is returned the operation is allowed.
4. If no authorization engine was configured, then default authorization engine is created, which checks whether the caller has same credentials as service (self authorization)

2.2. Client-side authorization

Client side authorization is done as a part of the authentication handler. GSI Secure Message authentication does client-side authorization only after the operation is completed. This is done as a part of the web services client handler. The other two authentication schemes supported, GSI Secure Conversation and GSI Transport, authorize the server during the handshake, prior to the operation invocation.

The Transport Level Security protocol allows for authorization (an expected DN comparison) during the handshake. This is disabled by default in the toolkit, unless delegation of credential is requested. If no delegation is requested, the configured authorization mechanism is invoked after the handshake is complete, prior to the operation invocation. If delegation is requested, authorization (expected DN comparison) is done during key exchange as a part of the protocol.

The toolkit supports self, gridmap, host and identity authorization on the client side. The authorization to be used is set programmatically on the Stub and the handler enforces it.

Chapter 6. APIs

Documentation for these interfaces can be found [here](#)¹.

1. Authorization Programming Model

Independent authorization settings can be configured on the server and client side. The security programming model on the server side is declarative and all configuration is done by setting a security descriptor. The descriptor can be a resource, service or container descriptor, depending on the required scope for the authorization. Alternatively the security settings can be done using programmatic security descriptor constructs. The client side the configuration is done by setting required properties on the stub used to make the method invocation. The security properties, and hence the authorization settings, can be set directly as properties on the stub, or a client security descriptor that encapsulates the individual properties may be written and in turn passed to the framework via a property on the stub object.

2. Authentication/message protection Programming Model

The authentication programming model differs between the client and server side. The client side model is programmatic in nature, i.e. security-related code is driven by making actual function calls, whereas the server-side model is declarative, i.e. security-related settings are declared in a security descriptor. For more information on the available client side calls see [Configuring client authentication and message/transport security](#). More information about the security descriptor can be found in [Java WS A&A Security Descriptor Framework](#).

3. API

- Generic Java Authorization Engine API
 - `org.globus.security.authorization.PDP`
 - `org.globus.security.authorization.PIP`
 - `org.globus.security.authorization.ChainConfig`
 - `org.globus.security.authorization.Interceptor`
- Stable API
 - `org.globus.wsrf.security.Constants`
 - `org.globus.wsrf.security.SecureResource`
 - `org.globus.wsrf.security.SecurityManager`
 - `org.globus.wsrf.security.authorization.Constants`
 - `org.globus.wsrf.security.impl.authorization.Authorization`
- Less stable API

¹ http://www.globus.org/api/javadoc-4.2.0/globus_java_ws_core

- `org.globus.wsrfl.impl.security.descriptor.ClientSecurityDescriptor`
- `org.globus.wsrfl.impl.security.descriptor.ServiceSecurityDescriptor`
- `org.globus.wsrfl.impl.security.descriptor.ResourceSecurityDescriptor`

DRAFT

Chapter 7. Services and WSDL

1. Secure Conversation Service

1.1. Protocol overview

This service provides a mechanism for generating a security session, i.e the negotiation of a shared secret which may be used to secure a set of subsequent messages. It is based on the [WS-Trust](#)¹ and [WS-SecureConversation](#)² specifications.

1.2. Operations

- `RequestSecurityToken`: This operation initiates a new security session negotiation. Furthermore, since the actual schema for this message is not unambiguously defined by the specifications, this is the actual schema used:

```
<xs:element name='RequestSecurityToken'>
  <xs:complexType name='RequestSecurityTokenType'>
    <xs:sequence>
      <xs:element ref='wst:TokenType' />
      <xs:element ref='wst:RequestType' />
      <xs:element ref='wst:BinaryExchange' />
    </xs:sequence>
    <xs:attribute name='Context' type='xs:anyURI' />
  </xs:complexType>
</xs:element>

<xs:element name='RequestSecurityTokenResponse'>
  <xs:complexType name='RequestSecurityTokenResponseType'>
    <xs:sequence>
      <xs:element ref='wst:TokenType' />
      <xs:element ref='wst:RequestType' />
      <xs:element ref='wst:BinaryExchange' />
    </xs:sequence>
    <xs:attribute name='Context' type='xs:anyURI' />
  </xs:complexType>
</xs:element>
```

- `RequestSecurityTokenResponse`: This operation continues a security session negotiation. Furthermore, since the actual schema for this message is not unambiguously defined by the specifications, this is the actual schema used:

```
<xs:element name='RequestSecurityTokenResponse'>
  <xs:complexType name='RequestSecurityTokenResponseType'>
    <xs:sequence>
      <xs:element ref='wst:TokenType' />
      <xs:element ref='wst:RequestType' />
      <xs:element ref='wst:BinaryExchange' />
    </xs:sequence>
    <xs:attribute name='Context' type='xs:anyURI' />
  </xs:complexType>
</xs:element>
```

¹ <http://www.ibm.com/developerworks/library/ws-trust/>

² <http://www-106.ibm.com/developerworks/library/ws-secon/>

```
</xs:complexType>
</xs:element>

<xs:element name='RequestSecurityTokenResponse'>
  <xs:complexType name='RequestSecurityTokenResponseType'>
    <xs:sequence>
      <xs:element ref='wst:TokenType' />
      <xs:element ref='wst:RequestType' />
      <xs:element ref='wst:BinaryExchange'
        minOccurs="0" />
      <xs:element ref='wsc:SecurityContextToken' />
    </xs:sequence>
    <xs:attribute name='Context' type='xs:anyURI' />
  </xs:complexType>
</xs:element>
```

In the above schema, the second RequestSecurityTokenResponse element refers to the final message in the exchange.

1.3. Resource properties

This service has no associated resource properties.

1.4. Faults

Both RequestSecurityToken and RequestSecurityTokenResponse throw the following faults:

- **ValueTypeNotSupportedFault:** This fault indicates that the value type attribute on the binary exchange token element is not supported by the service.
- **EncodingTypeNotSupportedFault:** This fault indicates that the encoding type attribute on the binary exchange token element is not supported by the service.
- **RequestTypeNotSupportedFault:** This fault indicates that the request type specified in the request type element is not supported by the service.
- **TokenTypeNotSupportedFault:** This fault indicates that the token type specified in the token type element is not supported by the service.
- **MalformedMessageFault:** This fault indicates that the message content received by the service does not conform to the expected content. This is necessary since the schema does not give a well defined content model.
- **BinaryExchangeFault:** This fault indicates that a failure occurred during the in the underlying security constant responsible for the session negotiation.
- **InvalidContextIdFault:** This fault indicates that the context id passed in the message is not valid within the context of this service or negotiation.

1.5. WSDL and Schema Definitions

- [WS-Trust WSDL](#)³

³ <http://www-106.ibm.com/developerworks/library/specification/ws-trust/ws-trust.wsdl>

- [WS-Trust XSD](#)⁴
- [WS-SecureConversation XSD](#)⁵
- Secure Conversation WSDL [fixme - link]

2. SAML Authorization Callout

The authorization framework as such does not have a WSDL interface. On the other hand one of the authorization schemes in the toolkit, a callout to an authorization service compliant with the specification published by the [OGSA Authorization Working Group \(OGSA-AuthZ\)](#)⁶ requires a WSDL interface for the service. The callout makes a query on the configured authorization service, which returns an authorization decision.

2.1. Protocol overview

The authorization service takes a query as input and returns an authorization decision. The [Security Assertion Markup Language \(SAML\)](#)⁷ is used for expressing the query and the decision. If any fault occurs, it is embedded as a part of the decision. The decision can be a permit, deny or indeterminate.

2.2. Operations

- `SAMLRequest`: Used to send queries to the authorization service, which after processing returns an authorization decision. All faults are embedded as part of the decision that is returned, i.e. no fault is declared at the WSDL level.
- `GetResourceProperty`: Gets the value of a specific resource property. This operation throws the following faults:
 - `ResourceUnknownFault`
 - `InvalidResourcePropertyQNameFault`
- `SetResourceProperties`: Sets the value for resource properties. This operation throws the following faults:
 - `ResourceUnknownFault`
 - `InvalidSetResourcePropertiesRequestContentFault`
 - `UnableToModifyResourcePropertyFault`
 - `InvalidResourcePropertyQNameFault`
 - `SetResourcePropertyRequestFailedFault`
- `QueryResourceProperties`: Used for the querying of resource properties using a query expression. This operation throws the following faults:
 - `ResourceUnknownFault`
 - `InvalidResourcePropertyQNameFault`

⁴ <http://www-106.ibm.com/developerworks/library/specification/ws-trust/ws-trust.xsd>

⁵ <http://www-106.ibm.com/developerworks/library/specification/ws-secon/ws-secureconversation.xsd>

⁶ <https://forge.gridforum.org/projects/ogsa-authz>

⁷ http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

- `UnknownQueryExpressionDialectFault`
- `InvalidQueryExpressionFault`
- `QueryEvaluationErrorFault`

2.3. Resource properties

- `supportedPolicies`: Contains identifiers for any or all access control policies that the authorization service is capable of rendering decisions regarding.
- `supportsIndeterminate`: Indicates whether the authorization service may return an "indeterminate" authorization decision. If set to false, only permit or deny is returned.
- `signatureCapable`: Indicates if the authorization service is capable of signing the decision returned. If not, only unsigned decisions are returned.

2.4. Faults

- `ResourceUnknownFault`⁸
- `InvalidSetResourcePropertiesRequestContentFault`⁹
- `UnableToModifyResourcePropertyFault`¹⁰
- `InvalidResourcePropertyQNameFault`¹¹
- `SetResourcePropertyRequestFailedFault`¹²
- `UnknownQueryExpressionDialectFault`¹³
- `InvalidQueryExpressionFault`¹⁴
- `QueryEvaluationErrorFault`¹⁵

2.5. WSDL and Schema Definition

- `OGSA-AuthZ Authorization Service WSDL`¹⁶

⁸ <http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf>

⁹ <http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf>

¹⁰ <http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf>

¹¹ <http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf>

¹² <http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf>

¹³ <http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf>

¹⁴ <http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf>

¹⁵ <http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf>

¹⁶ http://viewcvs.globus.org/viewcvs.cgi/wsrf/schema/core/security/authorization/authz_port_type.wsdl?rev=1.9&only_with_tag=globus_4_2_0&content-type=text/vnd.viewcvs-markup

Chapter 8. Framework-level Protocols

1. WS-Security

The framework implements the Web Services Security: SOAP Message Security¹, Web Services Security: Username Token Profile² and Web Services Security: X.509 Token Profile³ specifications.

2. Transport (HTTPS) Security

The transport security solution used by the framework consists of HTTP over SSL/TLS (HTTPS) using X.509 certificates. The path validation step has been augmented to support the Proxy Certificate Profile (RFC3820⁴).

¹ <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>

² <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf>

³ <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf>

⁴ <ftp://ftp.rfc-editor.org/in-notes/rfc3820.txt>

Chapter 9. Configuring client authentication and message/transport security

1. Interface introduction

Client-side security is set up by either setting individual properties on the `javax.xml.rpc.Stub` object used for the web service method invocation or by setting properties on a client-side security descriptor object, which in turn is propagated to client-side security handlers by making it available as a stub object property. Here are examples of the two approaches:

- Setting a property on the stub:

```
// Create endpoint reference
EndpointReferenceType endpoint = new EndpointReferenceType();
// Set address of service
String counterAddr =
    "http://localhost:8080/wsrf/services/CounterService";
// Get handle to port
CounterPortType port =
    locator.getCounterPortTypePort(endpoint);
// set client authorization to self
((Stub)port)._setProperty(Constants.AUTHORIZATION,
    SelfAuthorization.getInstance());
```

- Setting properties using a client descriptor:

```
// Client security descriptor file
String CLIENT_DESC =
    "org/globus/wsrf/samples/counter/client/client-security-config.xml";
// Create endpoint reference
EndpointReferenceType endpoint = new EndpointReferenceType();
// Set address of service
String counterAddr =
    "http://localhost:8080/wsrf/services/CounterService";
// Get handle to port
CounterPortType port =
    locator.getCounterPortTypePort(endpoint);
//Set descriptor on Stub
((Stub)port)._setProperty(Constants.CLIENT_DESCRIPTOR_FILE, CLIENT_DESC);
```

The descriptor file is described in detail in [Chapter 1, Security Descriptors Introduction](#).

**Note**

If the client needs to use transport security, the following API must be used to register the Axis transport handler for https:

```
import org.globus.axis.util.Util;
static {
    Util.registerTransport();
}
```

DRAFT

2. Syntax of the interface

DRAFT

Table 9.1. Client side security properties

Number	Task	Stub Configuration	De- Co- tion
1.	Allows for configuration of credentials for authentication.	Property: <code>org.globus.axis.gsi.GSIConstants.GSI_CREDENTIALS</code> Value equals the Instance of <code>org.ietf.jgss.GSSCredential</code> .	Sec- tion "C- ing tial
2.	Allows for configuring client-side authorization.	Property: <code>org.globus.wsrsecurity.Constants.AUTHORIZATION</code> Value equals the Instance of <code>org.globus.wsrsecurity.authorization.Authorization</code> If GSI Secure Transport or GSI Secure Conversation is used, the value should be an instance of <code>org.globus.gsi.gssapi.auth.Authorization</code> . But this translation is done automatically by the toolkit.	Rel- Sec- tion "C- ing atic ani
3.	Enable GSI Secure Conversation with specified message protection level.	1. Property: <code>org.globus.wsrsecurity.Constants.GSI_SEC_CONV</code> Values equal one of the following: <ul style="list-style-type: none"> • <code>Constants.ENCRYPTION</code> • <code>Constants.SIGNATURE</code> 2. Property: <code>org.globus.wsrsecurity.Constants.GSI_SEC_CONV_SECREPLY_UNNECESSARY</code> If the value is set to <code>Boolean.TRUE</code> , the GSI Secure conversation protection is not required in the reply message. By default, if the request was secured with GSI Secure Conversation, the response is also required to have the same protection. 3. Property: You can set the SOAP Actor of the GSI signed/encrypted SOAP message by using the <code>gssActor</code> property. We recommend that you <i>not</i> do this unless you <i>really</i> know what you are doing.	Rel- tion "C- ing cur ver

4.	Sets the GSI delegation mode. <i>Used for GSI Secure Conversation only.</i> If limited or full delegation is chosen, then some form of client-side authorization needs to be done (i.e client-side authorization cannot be set to none).	<p>Property:</p> <pre>org.globus.axis.gsi.GSIConstants.GSI_MODE</pre> <p>Value equals one of following:</p> <ol style="list-style-type: none"> 1. <code>GSIConstants.GSI_MODE_NO_DELEG</code>: No delegation is performed. 2. <code>GSIConstants.GSI_MODE_LIMITED_DELEG</code>: Limited delegation is performed. 3. <code>GSIConstants.GSI_MODE_FULL_DELEG</code>: Full delegation is performed. 	Rel tion "C ing cur ver
5.	Enables GSI Secure Transport with some protection level.	<p>Property:</p> <pre>org.globus.gsi.GSIConstants.GSI_TRANSPORT</pre> <p>Values equal one of the following:</p> <ul style="list-style-type: none"> • <code>Constants.ENCRYPTION</code> • <code>Constants.SIGNATURE</code> 	Rel tion "C ing cur por
6.	Enables anonymous authentication. <i>This option only applies to GSI Secure Conversation and GSI Transport.</i>	<p>Property:</p> <pre>org.globus.wsrp.security.Constants.GSI_ANONYMOUS</pre> <p>Value equals one of following:</p> <ol style="list-style-type: none"> 1. <code>Boolean.FALSE</code>: Anonymous authentication is disabled. 2. <code>Boolean.TRUE</code>: Anonymous authentication is enabled. 	Rel tion "C ing cur por

7.	Enable GSI Secure Message with specified message protection level.	<p>1. Property: <code>org.globus.wsrp.security.Constants.GSI_SEC_MSG</code></p> <p>Values equal one of the following:</p> <ul style="list-style-type: none"> • <code>Constants.ENCRIPTION</code> • <code>Constants.SIGNATURE</code> <p>2. Property: <code>org.globus.wsrp.security.Constants.GSI_SEC_MSG_SECREPLY_UNNECESSARY</code></p> <p>If the value is set to <code>Boolean.TRUE</code>, the GSI Secure Message protection is not required in the reply message. By default, if the request was secured with GSI Secure Message, the response is also required to have the same protection.</p> <p>3. Property: <code>org.globus.wsrp.security.Constants.GSI_SEC_MSG_SINGLECERT</code></p> <p>If the value is set to <code>Boolean.TRUE</code>, only a single certificate is used for the GSI Secure Message request. By default, the whole certificate chain is sent.</p> <p>4. Property:</p> <p>You can set the SOAP Actor of the signed message using the <code>x509Actor</code> property, but we do <i>not</i> recommend this unless you know what you are doing.</p>	Ret tion "C ing cur sag
8.	Enable WS-Security username/password authentication.	<p>Properties:</p> <p><code>org.globus.wsrp.security.Constants.USERNAME</code></p> <p>Value equals the username.</p> <p><code>org.globus.wsrp.security.Constants.PASSWORD</code></p> <p>Value equals the password.</p>	Ret tion "C ing nar wo

9.	Sets the credential that is used to encrypt the message (typically, the recipient's <i>public key</i>). <i>Used for GSI Secure Message only.</i>	<p>Property:</p> <pre>org.globus.wsrfl.impl.security.authentication .Constants.PEER_SUBJECT</pre> <p>Value equals the instance of <code>javax.security.auth.Subject</code>.</p> <p>The credential object needs to be wrapped in <code>org.globus.wsrfl.impl.security.authentication.encryption</code> and added to the set of public credentials of the Subject object.</p> <p>For example, if <code>publicKeyFilename</code> was the file that had the recipient's public key:</p> <pre>Subject subject = new Subject(); X509Certificate serverCert = CertUtil.loadCertificate(publicKeyFilename); EncryptionCredentials encryptionCreds = new EncryptionCredentials(new X509Certificate[] { serverCert }); subject.getPublicCredentials().add(encryptionCreds); stub._setProperty(Constants.PEER_SUBJECT, subject);</pre>	Re tion "C ing cur sag
10.	Sets the trusted certificates location.	<p>Property:</p> <pre>org.globus.wsrfl.security.TRUSTED_CERTIFICATES</pre> <p>Value should be a comma-separated list of directories and file names.</p>	Re tion "C ing cre " -
11.	Sets the SAML Authorization Assertion to embed in SOAP Header.	<p>Property:</p> <pre>org.globus.wsrfl.impl.security.authentication.Constants.SAML_AUTHZ_ASSERTION</pre> <p>Value should be an instance of <code>org.opensaml.SAMLAssertion</code>.</p>	Car con usi des

Chapter 10. Authorization domain-level interface

1. Interface introduction

Configuration on the server side is done using [Chapter 1, Security Descriptors Introduction](#). Make sure you have read about security descriptors (in the aforementioned link) before continuing with this chapter. Custom authorization mechanisms can be written and used as a part of the GT framework. The next section describes the steps involved.

On the client side, in addition to the security descriptor, properties on the stub can be set to configure security properties. Properties and values are described in detail in the next section.

2. Syntax of the interface

2.1. Configuring client-side authorization on the stub

The property to use depends on the authentication scheme:

- **If GSI Secure Transport or GSI Secure Conversation is used**, the `org.globus.axis.gsi.GSIConstants.GSI_AUTHORIZATION` property must be set on the stub. The value of this property must be an instance of an object that extends from `org.globus.gsi.gssapi.auth.GSSAuthorization`. All distributed authorization schemes have implementation in `org.globus.gsi.gssapi.auth` package.
- **For all other authentication schemes**, the `org.globus.wsrfl.impl.security.Constants.AUTHORIZATION` property must be set on the stub. The value of this property must be an instance of an object that implements the `org.globus.wsrfl.impl.security.authorization.Authorization` interface.
- Example:

```
// Create endpoint reference EndpointReferenceType
endpoint = new EndpointReferenceType();
// Set address of service
String counterAddr =
    "http://localhost:8080/wsrfl/services/CounterService";
// Get handle to stub object
CounterPortType port =
    locator.getCounterPortTypePort(endpoint);
// set client authorization to self
((Stub)port)._setProperty(Constants.AUTHORIZATION,
    SelfAuthorization.getInstance());
```

2.2. Writing custom client-side authorization scheme

Other than the distributed client authorization scheme, custom client-side authorization schemes can be written and can be set as the value for the appropriate property on the stub.

 **Note**

Security descriptors cannot be used to configure custom authorization schemes on the client side.

- **If the authentication scheme to be used is GSI Secure Transport or GSI Secure Conversation**, the custom authorization scheme should extend from `org.globus.gsi.gssapi.auth.GSSAuthorization`.

```
public class TestAuthorization extends GSSAuthorization {  
  
    // Provide some way to instantiate this class. Can use constructor  
    // with arguments to pass in parameters.  
    public TestAuthorization() {  
  
    }  
  
    public GSSName getExpectedName(GSSCredential cred, String host)  
        throws GSSException {  
  
        // Return the expected GSSName of the remote entity.  
    }  
  
    public void authorize(GSSContext context, String host)  
        throws AuthorizationException {  
  
        // Perform the authorization steps.  
        // context.getSrcName() provides the local GSSName  
        // context.getTargName() provides the remote GSSName  
  
        // if authorization fails, throw AuthorizationException  
    }  
}
```

The following describes the steps done for client side authorization during context establishment:

- Prior to initialization of context establishment the relevant handler (HTTPSSender in case of GSI Secure Transport or SecContextHandler in case of GSI Secure Conversation), invokes `getExpectedName` on the instance of `GSSAuthorization` set on the Stub.
- During context establishment, if the expected target name from previous step is not null, it is compared with the remote peer's `GSSName`. If it is not a match, context establishment is abandoned and an error is thrown.

If the expected target name is null, then a match is not done, unless the option of delegation is used. That is, if GSI Secure Conversation with delegation is used, then the expected target name cannot be null and must match the remote peer's identity.

- Once the context has been established, the `authorize` method is invoked.

 **Note**

Client authorization is done prior to invocation.

To configure the custom authorization scheme:

```
((Stub)port)._setProperty(GSIConstants.GSI_AUTHORIZATION,  
    new TestAuthorization());
```

Various authorization scheme implementations in package `org.globus.gsi.gssapi.auth` would serve as examples. [View CVS Link](#)¹

- **For all authentication schemes other than those in previous step** the `org.globus.wsrfl.impl.security.Constants.AUTHORIZATION` property must be set on the stub. The value of this property must be an instance of an object that implements the `org.globus.wsrfl.impl.security.authorization.Authorization` interface.

```
public class TestAuthorization implements Authorization {  
  
    // Provide some way to instantiate this class. Can use constructor  
    // with arguments to pass in parameters.  
    public TestAuthorization() {  
  
    }  
  
    public GSSName getName(MessageContext ctx)  
        throws GSSException {  
  
        // Return the expected GSSName of the remote entity.  
    }  
  
    void authorize(Subject peerSubject, MessageContext context)  
        throws AuthorizationException {  
  
        // Perform the authorization steps.  
        // peerSubject provides the remote Subject  
        // Use SecurityManager API to get local Subject  
  
        // if authorization fails, throw AuthorizationException  
    }  
}
```

The following describes the steps done for client side authorization:

- The client side handler `WSSecurityClientHandler`, invokes `authorize` method on the authorization instance.



Note

Client authorization is done after the invocation.

To configure the custom authorization scheme:

¹ <http://viewcvs.globus.org/viewcvs.cgi/jglobus/src/org/globus/gsi/gssapi/auth/?root=Java+COG&pathrev=HEAD>

```
((Stub)port)._setProperty(Constants.AUTHORIZATION,  
    new TestAuthorization());
```

Various authorization scheme implementations in package `org.globus.wsrfl.impl.security.authorization` would serve as examples. [View CVS Link](#)².

2.3. Writing a custom server-side authorization mechanism

The server side authorization framework can be configured to use custom authorization interceptors, bootstrap PIP, PIP and PDP. Detailed information on writing custom PDPs can be found in [GT Java Authorization Framework](#)³. Also, the section [Section 1, "Migrating Java WS Authorization Framework from GT 4.0"](#) describes migrating from older PDP/PIP implementations.

For example, a custom PDP must implement the interface `org.globus.security.authorization.PDP`.

Example:

```
package org.foobar;  
  
import ....;  
  
public class FooPDP implements PDP  
{  
    private Principal authorizedIdentity;  
  
    public Decision canAccess(RequestEntity requestEntity,  
        NonRequestEntity nonRequestEntity)  
        throws AuthorizationException {  
  
        // process and return decision  
    }  
  
    public Decision canAdminister(RequestEntity requestEntity,  
        NonRequestEntity nonRequestEntity)  
        throws AuthorizationException {  
  
        // process and return decision  
    }  
}
```

To use the above PDP one would configure a service security descriptor with the following authorization settings:

```
<securityConfig xmlns="http://www.globus.org">  
    ...  
    <auth value="fool:org.foobar.FooPDP"/>  
    ...  
</securityConfig/>
```

² <http://viewcvs.globus.org/viewcvs.cgi/wsrfl/java/core/source/src/org/globus/wsrfl/impl/security/authorization/?pathrev=HEAD>

³ [../gtJavaAuthEngine.pdf](#)

This security descriptor (identified as `.../foo-pdp-security-config.xml` below) can then be used by a service. The association is created by adding a couple of parameters to the service's WSDD entry:

```
...
<service name="MyDummyService"
  provider="Handler"
  style="document">
  ...
  <parameter name="securityDescriptor"
    value="/.../foo-pdp-security-config.xml"/>
  <parameter name="foo1-authorizedIdentity"
    value="/DC=org/DC=doe/OU=People/CN=John D"/>
  ...
</service>
```

Note that the parameter `<parameter>foo1-authorizedIdentity</parameter>` in the above configures the identity the PDP uses for authorizing incoming requests. The parameter name is derived by composing the prefix (`<parameter>foo1</parameter>`) used when specifying the PDP in the security descriptor with the property (`<parameter>authorizedIdentity</parameter>`) used in the PDP code.

Chapter 11. Configuring

Java WS A&A is configured using security descriptors. The following describes configuration settings specific for authorization and authentication. You can read the entire Java WS A&A Security Descriptor documentation [here](#).

- [Configuring authorization](#)
- [Configuring authentication/message protection](#)

1. Configuring authorization

1.1. Configuration overview

Security descriptors are mechanisms used to configure authorization mechanism and policy. The authorization on the server side can be configured at the container, service or resource level.

On the client side, authorization can be configured using security descriptors or as a property on the stub. This configuration can be done on a per invocation granularity

1.2. Server side authorization

The server side authorization can be configured at the container, service or resource level using

- Security descriptors using files. Refer to [Section 1, “Configuring Using Files”](#)
- Security descriptors programmatically. Refer to [Section 2, “Configuring Programmatically ”](#)

To write and configure a server-side custom authorization mechanism refer to [Section 2.3, “Writing a custom server-side authorization mechanism”](#).

1.3. Client side authorization

The client side authorization can be configured for each invocation.

- Security descriptors using files. Refer to [Section 1, “Configuring Using Files”](#), specifically [Section 1.2.2, “Configuring authorization mechanism ”](#).
- Security descriptors programmatically. Refer to [Section 2, “Configuring Programmatically ”](#)
- Properties on the Stub. Refer to [Section 2.1, “Configuring client-side authorization on the stub”](#)

To write and configure custom authorization mechanism refer to [Section 2.2, “Writing custom client-side authorization scheme”](#).

If no authorization mechanism has been specified, HostOrSelf authorization is used. In this scheme host authorization is tried first, if it fails, self authorization is attempted

2. Configuring authentication/message protection

2.1. Configuration overview

Configuration of service-side security settings can be achieved by using container or service security descriptor. Some of the security configuration, like the credential to use and trusted certificates location, can also be configured using CoG properties or rely on default location. **The preferred way is to provide these settings in a security descriptor.**

The next section provides details on the relevant properties. An overview of the syntax of security descriptors can be found in [Java WS A&A Security Descriptor Framework](#). Available CoG security properties can be found in [Chapter 2, Configuring](#)

2.2. Syntax of the interface

The following properties are relevant to authentication and message/transport security:

Table 11.1. Configuring server side authentication and message/transport security

Number	Task	Descriptor Configuration	Alternate Configuration
1	Credentials	Container or service descriptor configuration	<ul style="list-style-type: none"> • X509_USER_CERT or CoG Configuration: User certificate configuration • X509_USER_KEY or CoG Configuration: User key configuration • X509_USER_PROXY or CoG Configuration: User proxy configuration <p>If no explicit configuration is found, the default proxy is read from /tmp/x509_up_<uid>.</p>
2	Trusted Certificates	Container security descriptor configuration	CoG Configuration
3	Limited proxy policy configuration	Container or service descriptor configuration	None.
4	Replay Attack Window	Container or service descriptor configuration	None.
5	Replay Attack Filter	Container or service descriptor configuration	None.
6	Replay timer interval	Container descriptor configuration	None.
7	Context timer interval	Container descriptor configuration	None.

Chapter 12. Environment variable interface

1. Environmental variables for WS Authentication & Authorization (Java)

Refer to [Chapter 2, Configuring](#) for environment variables. Note that the above environment variables do *not* supersede any settings provided in security descriptors.

DRAFT

Chapter 13. PDP Reference

1. Introduction

[fixme - what are PDPs?]

If you have a PDP you'd like to contribute to the Globus Toolkit, use the following template:

- [PDP template](#)¹



Note

The above files are in DocBook XML format. Simply click the link, save to your hard drive, edit the file in a text or xml editor and email to ?. Don't worry about getting the tags right, it's enough to enter the information where it makes sense and we'll clean up the tags where necessary.

2. Access Control List PDP

2.1. Class name

```
org.globus.wsrfl.impl.security.authorizationnew.AccessControlListPDP
```

2.2. Overview

The PDP uses configured access control lists to ascertain if a subject can access an operation.

2.3. Configuration

`accessConfigFile` Property to configure the file containing policy for accessing resource. If not configured, file `access-acl.conf` is used by default.

`adminConfigFile` Property to configure file containing policy for administrating the resource. If not configured, file `admin-acl.conf` is used by default.

The PDP looks for the files in the following order:

1. Current directory (.)
2. `/etc`
3. `/etc/grid-security`
4. `$GLOBUS_LOCATION`

The configured files should have the following format:

```
subjectName=serviceName#method1,method2;testService2#method1,method2
```

¹ `../Java_WS_Security_PDP_Template.xml`

- Each subject DN should be a separate line. Equal to signs (=) and spaces should be escaped with backslash (\)
- An equal to must separate the subject DN and the service/operation name description.
- List of service URL and method names separated by semi-colon (;)
- Each service and method description should contain service URL followed by hash sign (#) and list of methods the subject DN is allowed to access/admin.
- Method names for each service separated by comma (,).

2.4. Decision Table

Bad configuration	INDETERMINATE
No configuration	INDETERMINATE
No policy for subject	DENY
Anonymous Access	DENY
No policy for subject to access service/operation	DENY
Policy exists for subject to access service/operation	PERMIT

2.5. Related interceptors

No interceptors are related other than the default bootstrap interceptor (X509 Bootstrap PIP), which is required to be used with this PDP.

3. GridMapAuthorization

3.1. Class name

```
org.globus.wsrfl.impl.security.authorization.GridMapAuthorization
```

3.2. Overview

This scheme checks the presence of a user's DN in a configured gridmap file. The grid map file contains a list of mappings from a user's DN to set of local user names that the user is mapped to. The DN and the list of comma-separated mappings are separated by a space. Each mapping is a separate new line by itself.

If the user is present in the configured gridmap file, the mappings are populated in the peer subject object as principals and the user is allowed access to the resource. Absence of DN in gridmap file yields a deny.

3.3. Configuration

The gridmap file location or an instance of GridMap object needs to be configured as a parameter to this PDP. The parameter names are:

`gridmap-file` Property to configure the location of the gridmap file. (From CoG add details on how this file is read in, relative to current directory ?)]

`gridmapObject` Property to configure an instance of GridMap class.

The PDP looks for the gridmap file in the following order:

1. GridMap object property in the configuration.
2. GridMap file property in the configuration.
3. GridMap object in the container default properties.
4. GridMap file property in the container default properties.

A default grid map file can be configured as described in [Section 1, “Configuring Default GridMap Files”](#).

3.4. Decision Table

No peer subject	INDETERMINATE
Bad/No gridmap configuration	INDETERMINATE
Failed gridmap refresh	INDETERMINATE
Anonymous Access	DENY
No gridmap entry	DENY
Gridmap entry present	PERMIT

3.5. Related interceptors

No interceptors are related other than the default bootstrap interceptor ([X509 Bootstrap PIP](#)), which is required to be used with this PDP.

4. Host Authorization

4.1. Class name

```
org.globus.wsrfl.impl.security.authorization.HostAuthorization
```

4.2. Overview

PDP performs host-based authorization of the client, i.e expects the client to be running with host credential.

4.3. Configuration

`url` Property that should be configured with the URL of the client.

`service` Property that can be configured with the service, if service credentials are used rather than regular host credentials. By default the value is set to *host*.

4.4. Decision Table

No peer subject	INDETERMINATE
Bad configuration	INDETERMINATE
<code>url</code> property not configured	INDETERMINATE

Anonymous Access	DENY
Peer DN does not match expected DN	DENY
Peer DN matches expected DN	PERMIT

4.5. Related interceptors

No interceptors are related other than the default bootstrap interceptor ([X509 Bootstrap PIP](#)), which is required to be used with this PDP.

5. IdentityAuthorization

5.1. Class name

`org.globus.wsrfl.impl.security.authorization.IdentityAuthorization`

5.2. Overview

Compares the peer subject with a specific configured subject DN.

5.3. Configuration

`identity` Property that should be configured with the expected peer DN.

5.4. Decision Table

No peer subject	DENY
No local subject	INDETERMINATE
Peer subject does not match configured (expected) subject DN	DENY
Peer subject matches the configured(expected) subject DN	PERMIT

5.5. Related interceptors

No interceptors are related other than the default bootstrap interceptor ([X509 Bootstrap PIP](#)), which is required to be used with this PDP.

6. No Authorization

6.1. Class name

`org.globus.wsrfl.impl.security.authorization.NoAuthorization`

6.2. Overview

This PDP always returns a permit.

6.3. Configuration

None.

6.4. Decision Table

Always returns a permit	-
-------------------------	---

6.5. Related interceptors

No interceptors are related

7. ResourceProperties Authorization

7.1. Class name

`org.globus.wsrfl.impl.security.authorization.ResourcePropertiesPDP`

7.2. Overview

The PDP enforces a parameter based authorization policy on `GetResourceProperty`, `GetMultipleResourceProperties` and `SetResourceProperties`. `QueryResourceProperties` is not protected by this PDP and to prevent malicious access of RPs through that method, access to that method must be protected using other schemes. `GetMultipleResourceProperties` access is allowed only if policy allows user access to all the RPs queried.

7.3. Configuration

`get-rp-pdp-config` Property that should be configured with a file containing authorization policy for access to `GetResourceProperty` and `GetMultipleResourceProperties` method.

`set-rp-pdp-config` Property that should be configured with a file containing authorization policy for access to `SetResourceProperty` method.

The format of the configuration files should be as follows:

```
subjectDN=qName1 ; qName2 ; qName3
```

All equal to signs and space in Subject DN should be escaped using backslash.

For example, if the set resource properties policy is defined as follows:

```
/C\=US/O\=Globus\ Alliance/OU\=User/CN\=101497d3dcd.3dcd5aef=\
{http://www.globus.org/tests/security}booleanVal ; {http://www.globus.org\
/tests/security}intVal1
```

[Above should be a single line without any spaces. Spaces provided here for document formatting] This implies that subject DN `/C=US/O=Globus Alliance/OU=User/CN=101497d3dcd.3dcd5aef` can set value of the RPs `{http://www.globus.org/tests/security}booleanVal` and `{http://www.globus.org/tests/security}intVal1`

7.4. Decision Table

Policy for set RP and get RP not configured	Initialize Exception
Erroneous configuration file	INDETERMINATE
Missing parameter value attribute	INDETERMINATE
No policy for user to access requested resource property.	DENY
Policy found for user to access requested resource property..	PERMIT

7.5. Related interceptors

- Default bootstrap interceptor ([X509 Bootstrap PIP](#)), is required to use this PDP.
- Parameter PIP ([Parameter PIP](#)) is required to use with this PDP to be able to collect information about the requested resource property. The ParameterPIP needs to be configured with the following :

```
servicePath getMultipleResourceProperties{http://docs.oasis-open.org\
/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd}\
getMultipleResourceProperties
```

```
servicePath getResourceProperty {http://docs.oasis-open.org/wsrf/2004/06/wsrf\
-WS-ResourceProperties-1.2-draft-01.xsd}getResourceProperty
```

```
servicePath setResourceProperties {http://docs.oasis-open.org/wsrf/2004/06/wsrf\
-WS-ResourceProperties-1.2-draft-01.xsd}SetResourceProperties
```

[Above should be a single line without any spaces. Spaces provided here for document formatting]

The servicePath needs to be replaced with the service endpoint that required resource property access to be authorized based on parameters.

8. SAML Authorization Callout

8.1. Class name

```
org.globus.wsrf.impl.security.authorization.SAMLAuthorizationCallout
```

8.2. Overview

This PDP can be used to talk to any authorization service that implements the [OGSA AuthZ²](#) interface. The steps involved include:

1. Secure call is made to the authorization service
2. Secure response is expected
3. Identity of soap message signature or tls connection is established
4. If response is signed, identity of response signature is established.

² <https://forge.gridforum.org/projects/ogsa-authz>

5. Issuer identity is determined from authz service is 4 or 5, in that order
6. If there are any errors constructing request, contacting authz service or parsing response, an indeterminate decision is issued by container
7. Response signature is verified.
8. Every assertion in the response is verified to be issued by issuer identity (point 5)
9. If any of the statement is not a permit on the particular subject, resource, action, a deny is returned by issuer identity. Otherwise a permit is returned.
10. In the future, a delegation step from the issuer identity (in 5) to some other issuers could be set as policy on SAMLAuthzCallout to establish a chain.

8.3. Configuration

<code>authzService</code>	Property that points to the authorization service to contact. The value should be a URL.
<code>securityMechanism</code>	Property that indicates the security mechanism to use to contact authorization service. Should be either "msg" for GSI Secure Message or "conv" for GSI Secure Conversation. If authorization service URL container "https" as protocol, GSI Secure Transport is used to contact the authorization service.
<code>protectionLevel</code>	Property that indicates the protection level to use to contact the authorization service. Should be either "sig" for signature and "enc" for encryption.
<code>authzServiceCertificate</code>	Property that points to the authorization service public certificate file. This property is required only if "securityMechanism" is GSI Secure Message and "protectionLevel" is encryption.
<code>authzServiceIdentity</code>	Property that has the expected identity of the authorization service. This is used for authorizing the call to authorization service.
<code>samlAuthzSimpleDecision</code>	Property indicates if SimpleAuthorizationDecisionStatement as defined in OGSA AuthZ specification is being requested from the authorization service.

8.4. Decision Table

Error constructing local SAML data types.	INDETERMINATE
Error converting EPR to string	INDETERMINATE
Error signing SAML Request	INDETERMINATE
Error accessing configured authorization service	INDETERMINATE
Null response from configured authorization service	INDETERMINATE
SAML exception from configured authorization service	INDETERMINATE
SAML Response signed by identity different from expected identity of configured authorization service	INDETERMINATE
If decision returned from service is anything other than permit.	DENY
If decision returned from service is permit.	PERMIT

8.5. Related interceptors

No interceptors are related other than the default bootstrap interceptor ([X509 Bootstrap PIP](#)), which is required to be used with this PDP.

9. SAML Authorization Assertion PDP

9.1. Class name

```
org.globus.wsrfl.impl.security.authorization.SAMLAuthzAssertionPDP
```

9.2. Overview

This PDP parses and enforces any SAML Authorization Decision Statements that are a part of the requester's attribute bag. Typically SAMLAuthzAssertionPIP is used in tandem with this PDP to collect those attributes.

The PDP expects the service name as SAML Resource and the operation name as action.

9.3. Configuration

No configuration is required.

9.4. Decision Table

No requester attributes	NOT_APPLICABLE
No resource attributes	INDETERMINATE
No action attributes	INDETERMINATE
If atleast one of the SAML Decision statement is permit for access to said resource and action	PERMIT
If none of the SAML Decision statement is permit for access to said resource and action	DENY

9.5. Related interceptors

- Default bootstrap interceptor ([X509 Bootstrap PIP](#)), is required to use this PDP.
- SAML Authorization Assertion PIP ([SAMLAuthzAssertPIP](#)) is required to use with this PDP to be able to collect SAML Authorization Assertions that are sent in as a part of the request.

10. Self Authorization

10.1. Class name

```
org.globus.wsrfl.impl.security.authorization.SelfAuthorization
```

10.2. Overview

This PDP does self authorization and expects the peer subject to match the local subject, which is the current JAAS Subject associated with the service/resource. The current JAAS subject is determined by the value of the run-as element in the service security descriptor (see [Configuring run-as mode](#)).

10.3. Configuration

No configuration is required

10.4. Decision Table

No peer subject	DENY
No local subject	INDETERMINATE
Peer subject matches local subject	PERMIT
Peer subject does not match local subject	DENY

10.5. Related interceptors

No interceptors are related other than the default bootstrap interceptor ([X509 Bootstrap PIP](#)), which is required to be used with this PDP.

11. Username Authorization

11.1. Class name

```
org.globus.wsrfl.impl.security.authorization.UsernameAuthorization
```

11.2. Overview

Uses Java Login module to authorize based on user name and password used on the method call. The username and password are passed to the Login module using `javax.security.auth.callback.NameCallback` and `javax.security.auth.callback.PasswordCallback`.

Username authorization as part of the authorization framework for a service involves setting up a Configuration class that reads in the Login Module to be using. It is also possible to specify the login modules to be used by altering some parameters in the `java.security` file located at `$JAVA_HOME/jre/lib/security` which has been discussed under the general configuration section. Steps for username authorization using Login Modules configured via custom Configuration classes is described under the custom configuration section.

11.3. Configuration

Login modules can be setup for use by our application in two different ways. One, by adding the login module to the list of login modules used by the JVM. This method has been explained under the JVM Configuration. Second, by creating a custom configuration class to dynamically pick up the login module.

11.3.1. JVM Configuration

Some login module and configuration file needs to be configured (example: jre/lib/ext/jaasmod.jar and jre/lib/security/java.security). A sample configuration entry:

```
login.configuration.provider=com.sun.security.auth.login.ConfigFile
login.config.url.1=file:${java.home}/lib/security/jaas.config
```

Sample jaas.config file:

```
Login {
  com.sun.security.auth.module.NTLoginModule required;
};
```

11.3.2. Custom Configuration

A configuration class is used to setup a specified login module for executing authorization. The class details are provided to the Toolkit through the security descriptor for the service that enforces username authorization. In order for the username authorization handler to use a custom Login Configuration, a parameter with name "login-config" and a value containing the custom Configuration classname has to be specified along with the PDP in the security descriptor. The PDP in this case is "Username Authorization". The relevant snippet from the service security descriptor is shown below:

```
<authzChain>
<pdp>
  <interceptor name="prefix:org.globus.wsrfl.impl.security.authorization.UsernameAuthzChain">
    <parameter>
      <param:nameValueParam>
        <param:parameter name="login-config" value="org.globus.wsrfl.impl.security.authorization.UsernameAuthzChain">
        </param:nameValueParam>
      </parameter>
    </interceptor>
  </pdp>
</authzChain>
```

Internal Details

The usernameAuthorization PDP loads the Configuration specified in the parameter "login-config" and sets that as the current configuration. (These are internal details which may be disregarded safely for the purposes of writing up a custom configuration) `org.globus.wsrfl.impl.security.authorization.UsernameAuthorization`

```
String className = (String) this.chainConfig.getProperty(this.prefix, "login-config");
Class config = ClassLoaderUtils.forName(className);
Object loginConfig = config.newInstance();
if(loginConfig instanceof Configuration){
  Configuration.setConfiguration((Configuration) loginConfig);
}
```

The custom Configuration class has to be a subclass of `javax.security.auth.login.Configuration`. An example custom Configuration class is available in the unit tests directory at `org.globus.wsrfl.impl.security.authorization.LoginConfiguration`. The login module that has to be used is specified using a `login.config` file that the configuration class uses. The configuration class loads the appropriate properties from this file. A sample entry for this file is:

```
Login {
    org.globus.wsrfl.impl.security.authorization.UsernameLoginModule required;
};
```

More details regarding Login Configuration files can be found [here](#)³. The next section focuses on the setup and usage of login modules.

11.4. Login Modules

The login modules can be used to handle the authorization of users. The details regarding writing login modules is beyond the scope of this document and can be found at in the [Login Module Developers' guide](#)⁴

The unit tests have an example of a login module `org.globus.impl.security.authorization.UsernameLoginModule`. The username and password to authorize the client have to be passed via the client security descriptor similar to the one listed below. If the password type is not provided then it defaults to `PasswordText`. In that case, the passwords are sent unencrypted over the wire.

```
<clientSecurityConfig xmlns="http://www.globus.org/security/descriptor/client">
  <GSISecureTransport>
    <integrity/>
  </GSISecureTransport>

  <usernameType>
    <username value="globus"/>
    <passwordType>
      <password value="unittesting"/>
      <type value="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-authentication/>
    </passwordType>
  </usernameType>

  <authz value="none"/>

</clientSecurityConfig>
```

This login module handles the authorization of usernames and passwords supplied either as Text (unencrypted format) or as a Digest (encrypted format.) by comparing them to the username and password stored on the server in a file. The format of the parameters in the password file used by our example Login Module is illustrated in the following example.

```
~/tests/unit/etc/Test-authz-pwd
```

```
name=globus
```

³ <http://java.sun.com/j2se/1.4.2/docs/guide/security/jaas/JAASRefGuide.html#AppendixB>

⁴ <http://java.sun.com/j2se/1.4.2/docs/guide/security/jaas/JAASLMDevGuide.html#Intro>

```
password=unittesting  
usernames=globus
```

In summary, Login Modules can be used to implement username authorization. This can be done by either changing the java environment to invoke your login module or by programmatically setting the authorization framework to use your login module. The unit tests provide a good example of how this can be done programmatically which in most cases would prove beneficial.

11.5. Decision Table

If no error is returned from Login module	PERMIT
---	--------

11.6. Related interceptors

None.

DRAFT

Chapter 14. PIP Reference

1. Introduction

[fixme - what are PIPs]

If you have a PIP you'd like to contribute to the Globus Toolkit, use the following template:

- [PIP template](#)¹



Note

The above files are in DocBook XML format. Simply save the link to your hard drive, edit the file in a text or xml editor and email to ?. Don't worry about getting the tags exactly right, it's enough to enter the information where it makes sense and we'll clean up the tags where necessary.

2. Container PIP

2.1. Class name

```
org.globus.wsrfl.impl.security.authorization.ContainerPIP
```

2.2. Overview

This implements the BootstrapPIP interface [PIP-glossary] and is used with in the toolkit to initialize the request entities. It collects information about the service and operation invoked. It is always invoked prior to any authorization processing.

2.3. Configuration

No configuration is required.

2.4. Attributes Collected

This PIP collects three attributes described in the following tables:

¹ ../Java_WS_Security_PIP_Template.xml

Table 14.1. Attribute I

Description of attribute	Message Context associated with the thread
Identity attribute	Identity attribute
Attribute ID	Constants.MSG_CTX_ATTRIBUTE_URI
Datatype	Constants.MSG_CTX_DATATYPE_URI
Issuer	null. The issuer is null since the message context is required to construct the container entity, which is the default issuer for attributes collected in the container.
Validity from	Current time
Validity to	Infinity

Table 14.2. Attribute II

Description of attribute	URL of the service invoked.
Identity attribute	Identity attribute
Attribute ID	Constants.SERVICE_ATTRIBUTE_ID_URI
Datatype	Constants.STRING_DATATYPE_URI
Issuer	Container
Validity from	Current time
Validity to	Infinity

Table 14.3. Attribute III

Description of attribute	Name of the operation invoked.
Identity attribute	Identity attribute
Attribute ID	Constants.OPERATION_ATTRIBUTE_ID_URI
Datatype	Constants.STRING_DATATYPE_URI
Issuer	Container
Validity from	Current time
Validity to	Infinity

This PIP also sets up the container issuer entity, which is used as the default issuer for attributes collected in the container. The entity has the following attributes:

Table 14.4. Attribute I

Description of attribute	Container id
Identity attribute	Identity attribute
Attribute ID	Constants.CONTAINER_ATTRIBUTE_URI
Datatype	Constants.STRING_DATATYPE_URI
Issuer	null
Validity from	Current time
Validity to	Infinity

Table 14.5. Attribute II

Description of attribute	Java Principals from container credential, only if credentials are configured.
Identity attribute	Identity attribute
Attribute ID	Constants.PRINCIPAL_ATTRIBUTE_ID
Datatype	Constants.PRINCIPAL_DATATYPE_URI
Issuer	null
Validity from	Current time
Validity to	Infinity

Table 14.6. Attribute III

Description of attribute	Java Subject from container credential, only if credentials are configured.
Identity attribute	Identity attribute
Attribute ID	Constants.SUBJECT_ATTRIBUTE_ID
Datatype	Constants.SUBJECT_DATATYPE_URI
Issuer	null
Validity from	Current time
Validity to	Infinity

The container entity is created with the same attributes as above with the above entity as the issuer.

2.5. Related interceptors

None.

3. X509Bootstrap

3.1. Class name

```
org.globus.wsrfl.impl.security.authorization.X509BootstrapPIP
```

3.2. Overview

This implements the BootstrapPIP interface [PIP-glossary] which is used when X509 Certificates are used during authentication scheme. It collects peer entities' attributes obtained from the certificates presented by the peer.

3.3. Configuration

No configuration is required.

3.4. Attributes Collected

This PIP collects two attributes described in the following tables:

Table 14.7. Attribute I

Description of attribute	Peer's Subject object
Identity attribute	Identity attribute
Attribute ID	Constants.SUBJECT_ATTRIBUTE_ID
Datatype	Constants.SUBJECT_DATATYPE_URI
Issuer	Container Issuer Entity
Validity from	Current time
Validity to	Infinity

Table 14.8. Attribute II

Description of attribute	Peer's principals
Identity attribute	Identity attribute
Attribute ID	Constants.PRINCIPAL_ATTRIBUTE_ID
Datatype	Constants.PRINCIPAL_DATATYPE_URI
Issuer	Container Issuer Entity
Validity from	Current time
Validity to	Infinity

3.5. Related interceptors

If X509 Certificates are used for authentication, this bootstrap is used by the Authorization Framework by default.

4. SAML Authorization Assertion PIP

4.1. Class name

```
org.globus.wsrfl.impl.security.authorization.SAMLAuthzAssertionPIP
```

4.2. Overview

The PIP extracts SAML Authorization Assertion from the request and adds it to the bag of attributes. The message context and the proxy certificate are checked to see if SAML Authorization Assertions are presents.

If the subject DN in the decision statement matches with the requestor's then the attribute is merged with the requestor's bag of attributes.

4.3. Configuration

No configuration information is required.

4.4. Attributes Collected

This PIP collects attributes described in the following tables:

Table 14.9. Attribute I

Description of attribute	Subject DN from the subject in SAML Authorization Decision Statement (one attribute per statement in assertion)
Identity attribute	Identity attribute
Attribute ID	Constants.PRINCIPAL_ATTRIBUTE_ID
Datatype	Constants.PRINCIPAL_DATATYPE_URI
Issuer	Container Issuer Entity
Validity from	Extracted from assertion
Validity to	Extracted from assertion

Table 14.10. Attribute II

Description of attribute	SAML Authoization Decision Statement (one attribute per statement in assertion)
Identity attribute	Non-Identity attribute
Attribute ID	Constants.SAML_AUTHZ_DECISION_ATTRIBUTE_ID
Datatype	Constants.SAML_AUTHZ_DECISION_DATA_TYPE
Issuer	Container Issuer Entity
Validity from	Extracted from assertion
Validity to	Extracted from assertion

4.5. Related interceptors

This PIP can be used in tandem with SAMLAuthzAssertionPDP.

5. Parameter PIP

5.1. Class name

```
org.globus.wsrfl.impl.security.authorization.ParameterPIP
```

5.2. Overview

This PIP extracts configured parameter element from the SOAPMessage. The parameter is added as an action attributes in the associated RequestAttribute.

5.3. Configuration

`parameterConfig` Property pointing to configuration file with information about the service, method and parameter to extract as attributes. If configured file name is not absolute, an attempt is made to find the file as provided, if not an attempt is made to locate it relative to `GLOBUS_LOCATION` and if that fails, an attempt it made to locate it relative to current directory.

The configuration file is read and stored as `SOAPPParameter`. This class is used to store a specific parameter element path for a given operation for a said service. `servicePath` `operationName` `ParameterPath`

The parameter path is a list of QNames, where each QName is QName of a child element of previous QName element. The parameterPath is a string with string representation of each QName, in the order it needs to be looked into with semicolon (;) as delimiter. For example, `{http://temp.ns}element1;{http://temp.ns}nextElem2;{http://temp.ns}nextElem3` would represent the parameter `{http://temp.ns}nextElem3`. The SOAPBody element here is `{http://temp.ns}element1`, with `nextElem2` as its child and `nextElem3` as its child.

5.4. Attributes Collected

This PIP collects two attributes described in the following tables:

Table 14.11. Attribute I

Description of attribute	Configured parameter if it occurs in that operation. The value is an object of type <code>org.w3c.dom.Node</code> and represents the parameter of the operation.
Identity attribute	Identity attribute
Attribute ID	Parameter path as described in previous section.
Datatype	<code>Constants.PARAMETER_PATH_DATA_TYPE</code>
Issuer	Container Issuer Entity
Validity from	Current time
Validity to	Infinity

5.5. Related interceptors

This PIP can be used in tandem with `ResourcePropertiesPDP` to parameter-based authorization for resource property access.

Chapter 15. Debugging

Because Java WS A&A is built on top of Java WS Core, developer debugging is the same as described in [Chapter 10, Debugging](#).

For information about system administrator logs, see [Chapter 6, Debugging](#).

Java WS Core also provides an API for CEDPs-compliant logging as described in [Section 1.2, “Configuring system administration logs”](#).

1. Debugging authorization

Log output from the authorization framework is a useful tool for debugging issues. Because the Authorization Framework is built on top of Java WS Core, developer debugging is the same as described in [Chapter 10, Debugging](#).

1.1. Development Logging in Java WS Core

The following information applies to Java WS Core and those services built on it.

Logging in the Java WS Core is based on the [Jakarta Commons Logging](#)¹ API. Commons Logging provides a consistent interface for instrumenting source code while at the same time allowing the user to plug-in a different logging implementation. Currently we use [Log4j](#)² as a logging implementation. Log4j uses a separate configuration file to configure itself. Please see Log4j documentation for details on the [configuration file format](#)³.

1.1.1. Configuring server side developer logs

Server side logging can be configured in `$GLOBUS_LOCATION/container-log4j.properties`, when the container is stand alone container. For tomcat level logging, refer to [Logging for Tomcat](#)⁴, . The logger `log4j.appender.A1` is used for developer logging and by default writes output to the system output. By default it is set for all warnings in the Globus Toolkit package to be displayed.

Additional logging can be enabled for a package by adding a new line to the configuration file. Example:

```
#for debug level logging from org.globus.package.FooClass
log4j.category.org.globus.package.name.FooClass=DEBUG
#for warnings from org.some.warn.package
log4j.category.org.some.warn.package=WARN
```

1.1.2. Configuring client side developer logs

Client side logging can be configured in `$GLOBUS_LOCATION/log4j.properties`. The logger `log4j.appender.A1` is used for developer logging and by default writes output to the system output. By default it is set for all warnings in the Globus Toolkit package to be displayed.

¹ <http://jakarta.apache.org/commons/logging/>

² <http://logging.apache.org/log4j/>

³ [http://logging.apache.org/log4j/docs/api/org/apache/log4j/PropertyConfigurator.html#doConfigure\(java.lang.String,org.apache.log4j.spi.LoggerRepository\)](http://logging.apache.org/log4j/docs/api/org/apache/log4j/PropertyConfigurator.html#doConfigure(java.lang.String,org.apache.log4j.spi.LoggerRepository))

⁴ <http://tomcat.apache.org/tomcat-5.5-doc/logging.html>

1.2. Enabling verbose logging

As described in the above section, configuration files need to be edited to enable logging at different levels. For example, to see all logging for server side authorization, the following lines need to be added to container logging configuration file. To see client-side authorization framework logging, the same line needs to be added to *\$GLOBUS_LOCA-TION/log4j.properties*.

```
log4j.category.org.globus.wsrfl.impl.security.authorization=DEBUG
```

The authorization module uses [Java CoG Kit](#)⁵ for some of the functionality. To turn on logging for that functionality, the following can be added to the relevant logging file, depending on whether it is the client or the server side logging.

```
log4j.category.org.globus.gsi.jaas=DEBUG
log4j.category.org.globus.gsi.gssapi=DEBUG
log4j.category.org.globus.security.gridmap=DEBUG
```

⁵ <http://www.globus.org/cog/java/>

Chapter 16. Troubleshooting

For a list of common errors in GT, see [Error Codes](#).

DRAFT

1. Error Messages For Java WS A&A

DRAFT

Table 16.1. Java WS A&A Errors

Error Code	Definition	Possible Solutions
[JWSSEC-248] Secure container requires valid credentials	This error occurs when <code>globus-start-container</code> is run without any valid credentials. Either a proxy certificate or service/host certificate needs to be configured for the container to start up.	<ol style="list-style-type: none"> <li data-bbox="805 300 1334 604">1. If you are not looking to start up a container that uses GSI Secure Transport, which is used by the container by default, use <code>globus-start-container -nosec</code>. You will be able to use insecure clients and services. However, this also implies that if you have not configured individual services with credentials, you will not be able to securely access the service. <li data-bbox="805 615 1334 804">2. If you are running a personal container, generate a proxy certificate with <code>grid-proxy-init</code>. If the proxy certificate is not in the default location, configure the container security descriptor as described in Configuring Container Security Descriptor. <li data-bbox="805 825 1334 930">3. If you want to use host certificates, configure the container security descriptor as described Configuring Credentials.
Failed to start container: Container failed to initialize [Caused by: [JWSSEC-250] Failed to load certificate/key file]	This error occurs if the file path to the container certificate and key configured are invalid.	<ol style="list-style-type: none"> <li data-bbox="805 963 1334 1163">1. The path to the container certificate and key are configured in <code>\$GLOBUS_LOCATION/etc/globus_wsrf_core/global_security_descriptor.xml</code>. This file is loaded as described [here - fixme link]. Ensure that the path is correct.
Failed to start container: Container failed to initialize [Caused by: [JWSSEC-249] Failed to load proxy file]	This error occurs if container proxy file configured is invalid.	<ol style="list-style-type: none"> <li data-bbox="805 1194 1334 1394">1. The path to the container proxy certificates are configured in <code>\$GLOBUS_LOCATION/etc/globus_wsrf_core/global_security_descriptor.xml</code>. This file is loaded as described [here - fixme link]. Ensure that the path is correct.
Failed to start container: Container failed to initialize [Caused by: [JWSSEC-245] Error parsing file: "etc/globus_wsrf_core/global_security_descriptor.xml" [Caused by: ...]	This error occurs if the container security descriptor configured is invalid.	<ol style="list-style-type: none"> <li data-bbox="805 1425 1334 1520">1. The container security descriptor should conform to the Container Security Descriptor Schema.¹ <li data-bbox="805 1541 1334 1604">2. Refer to the "Caused by: " section for details on the specific element that is not correct.

¹ http://www.globus.org/toolkit/docs/4.2.0/security/container_security_descriptor.xsd

Error Code	Definition	Possible Solutions
[JGLOBUS-77] Unknown CA	This error occurs if the CA certificate for the credentials being used is not installed correctly.	<ol style="list-style-type: none"><li data-bbox="805 243 1336 401">1. If this issue occurs on the server side, the container is not configured with CA certificates. The container looks for trusted certificates in the default location as described Java CoG Toolkit FAQ²<li data-bbox="805 428 1336 520">2. On the server side, the trusted certificates can be configured as described in Trusted Certificates<li data-bbox="805 548 1336 640">3. On the client side, trusted certificates can be configured as described in Configuring Trusted Credentials

² <http://www.globus.org/cog/distribution/1.2/FAQ.TXT>

Chapter 17. Related Documentation

See [Section 9, “Associated Standards”](#) for a list of associated standards.

DRAFT

Glossary

E

End Entity Certificate (EEC) A certificate belonging to a non-CA entity, e.g. you, me or the computer on your desk.

P

private key The private part of a key pair. Depending on the type of certificate the key corresponds to it may typically be found in `$HOME/.globus/userkey.pem` (for user certificates), `/etc/grid-security/hostkey.pem` (for host certificates) or `/etc/grid-security/<service>/<service>key.pem` (for service certificates).

For more information on possible private key locations see [this](#).

proxy certificate A short lived certificate issued using a EEC. A proxy certificate typically has the same effective subject as the EEC that issued it and can thus be used in its place. GSI uses proxy certificates for single sign on and delegation of rights to other entities.

For more information about types of proxy certificates and their compatibility in different versions of GT, see <http://dev.globus.org/wiki/Security/ProxyCertTypes>.

public key The public part of a key pair used for cryptographic operations (e.g. signing, encrypting).

T

transport-level security Uses transport-level security (TLS) mechanisms.

GT 4.2.0 Migrating Guide for Java WS A&A

Table of Contents

1. Migrating Java WS Authorization Framework from GT 4.0	1
2. Migrating Java WS Authorization Framework from GT3	3
3. Migrating host credentials from GT3 and GT2	3
Glossary	3

<titleabbrev>Migrating Guide</titleabbrev>

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

1. Migrating Java WS Authorization Framework from GT 4.0

Java WS Authorization framework has been reworked with WS independent authorization engine and separate authorization module. The following sections describe migrating from GT 4.0.x and intermediate GT 4.1.x development releases.

1.1. Interface and code changes

The Java WS Authorization Framework now uses the generic GT Java Authorization Framework, that eliminates dependency on web services components. The following changes will be needed from GT 4.0.x:

1. Package name: PDP/PIP interfaces, attribute processing classes and configuration classes are now used from the generic authorization engine. Hence the authorization interfaces have changed from `org.globus.wsrp.security.authorization` to `org.globus.security.authorization`. The following classes have changed:
 - Attribute
 - AttributeCollection
 - AttributeException
 - AttributeIdentifier
 - AuthorizationConfig
 - AuthorizationDeniedException
 - AuthorizationEngineSpi
 - AuthorizationException
 - BootstrapPIP
 - ChainConfig

- CloseException
 - Decision
 - EntityAttributes
 - IdentityAttributeCollection
 - InitializeException
 - Interceptor
 - InterceptorConfig
 - InterceptorException
 - PDP
 - PIP
2. RequestAttributes class: This class has been renamed as RequestEntities with no functionality change
 3. PIPResponse class: This class has been renamed as NonRequestEntities. Functionality from the older class has been preserved, with additional methods to merge attributes to this structure has been added.
 4. PIP Interface change: The new PIP interface is as follows:

```
public
    NonRequestEntities collectAttributes(RequestEntities requestAttr) throws AttributionException;
```

MessageContext has been removed from the interface. Refer to item (7) on information on retrieving message context.

This interface does not extend from Interceptor interface. But PIPInterceptor interface is equivalent to the previous version of the PIP interface, with collect attributes method and interceptor interface methods.

5. PDP Interface change: The new PDP interface is as follows:

```
public Decision
    canAccess(RequestEntities requestEntities, NonRequestEntities nonReqEntities) throws AuthorizationException;
    public Decision canAdminister(RequestEntities requestEntities, NonRequestEntities nonReqEntities) throws AuthorizationException;
```

NonRequestAttributes class encompasses the three List objects for non-request subject, resource and action.

The RequestAttributes class is replaced by RequestEntities class as is. MessageContext has been removed from the interface. Refer to item (7) on information on retrieving message context.

This interface does not extend from Interceptor interface. But PDPInterceptor interface is equivalent to the previous version of the PDP interface, with collect attributes method and interceptor interface methods.

6. Providers package: The providers that were a part of the authorization package are now a part of the generic interface. So the FirstApplicable and PermitOverride combining algorithm interface, in addition to the AbstractEngine class are now a part of the new package, org.globus.security.authorization.providers.

7. Message Context: ContainerPIP, the default PIP that is used by the GT framework to initialize request context, adds the message context associated with the request as an environment attribute with null issuer. To extract the message context, the following code snippet can be used:

```
RequestEntities reqEntities; org.apache.axis.MessageContext msgCtx =
    AttributeUtil.getMessageContext(reqEntities.getEnvironment(), null);
```

1.2. Authorization Module Changes

The Java WS server side authorization code has been moved to a separate module called `authorization`. A migration guide, that outlines the changes needed for services that build on Java WS Core, is provided [here](#).¹

2. Migrating Java WS Authorization Framework from GT3

While the GT4 version of this component has similar features to the GT3 version, some of the configuration methodology has changed and some features have been enhanced. Refer to [Section 1.4.5, “Configuring authorization mechanisms”](#) for changes in configuration.

3. Migrating host credentials from GT3 and GT2

GT2 and GT3 services were set up to run with root owned *host credentials*. In GT4 most, but not all, services will run as the **globus** user. To allow the **globus** user to start services using host credentials, the **globus** user needs to be able to access them. This requirement can be satisfied by making a copy of the root-owned host credentials, i.e. the *host certificate* and *private key*, owned by the **globus** user. In GT4 this copy is assumed to be `/etc/grid-security/container{cert,key}.pem`.

Glossary

H

host certificate An EEC belonging to a host. When using GSI this certificate is typically stored in `/etc/grid-security/hostcert.pem`. For more information on possible host certificate locations see the [GSI C Developer's Guide](#).

host credentials The combination of a host certificate and its corresponding private key.

P

private key The private part of a key pair. Depending on the type of certificate the key corresponds to it may typically be found in `$HOME/.globus/userkey.pem` (for user certificates), `/etc/grid-security/hostkey.pem` (for host certificates) or `/etc/grid-security/<service>/<service>key.pem` (for service certificates).

For more information on possible private key locations see [this](#).

¹ http://dev.globus.org/wiki/Java_WS_Core/Independent_Java_Authz_Module

GT4 Java WS A&A Quality Report

Table of Contents

1. Test coverage reports	1
2. Code analysis reports	1
3. Outstanding bugs	1
4. Bug Fixes	2
5. Performance reports	3

<titleabbrev>Quality Report</titleabbrev>

1. Test coverage reports

- [Clover report](#)¹

2. Code analysis reports

- [PMD report](#)²
- [FindBugs report](#)³

3. Outstanding bugs

- [Bug 2362](#):⁴ location of user proxy for java inconsistencies
- [Bug 2445](#):⁵ Holder problem
- [Bug 2907](#):⁶ Secure Conversation (Encryption) does not provide any message level security for the SOAP headers
- [Bug 3027](#):⁷ Kerberos based authentication option for GT4
- [Bug 3171](#):⁸ add RFC 2253 principal name to JAAS subject
- [Bug 3449](#):⁹ ERROR container.GSIServiceThread
- [Bug 3603](#):¹⁰ Remote exceptions thrown contain server specific information
- [Bug 3928](#):¹¹ IPv6 addresses in reverse lookups - fix or faq?

¹ <http://www.mcs.anl.gov/~gawor/javawscore/HEAD/clover/clover-reports-html/>

² http://www.mcs.anl.gov/~gawor/javawscore/HEAD/pmd/pmd_report.html

³ http://www.mcs.anl.gov/~gawor/javawscore/HEAD/findbugs/findbugs_report.html

⁴ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2362

⁵ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2445

⁶ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2907

⁷ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3027

⁸ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3171

⁹ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3449

¹⁰ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3603

¹¹ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3928

- [Bug 3941](#):¹² Expired credentials detected - candidate for sec error msg improvements
- [Bug 4222](#):¹³ Allow for credential refresh in subscriptions
- [Bug 4403](#):¹⁴ Secure calls for secure context establishment
- [Bug 4442](#):¹⁵ Security descriptor refresh
- [Bug 5008](#):¹⁶ voms-proxy-init creates non-critical KeyUsage extension which causes Java GSI to raise exception
- [Bug 5026](#):¹⁷ Signarure validation failure on GRAM/RFT interaction on some cases

4. Bug Fixes

- [Bug 2535](#):¹⁸ <proxy-file> causes container to fail
- [Bug 2651](#):¹⁹ /dev/random vs. /dev/urandom
- [Bug 2743](#):²⁰ grid-mapfile location should be in global security descriptor
- [Bug 2207](#):²¹ Missing security error 'timestampNotOk'
- [Bug 2651](#):²² /dev/random vs. /dev/urandom
- [Bug 2743](#):²³ grid-mapfile location should be in global security descriptor
- [Bug 2899](#):²⁴ relative path does not work for credentials in Security Descriptor
- [Bug 2900](#):²⁵ Job submssion does not work using relative path in global_security_descriptor.xml and absolute path in sudoers.
- [Bug 2955](#):²⁶ Job submission fails when container is started from non GLOBUS_LOCATION
- [Bug 2969](#):²⁷ Too relaxed rules on DN comparisons (all versions of GT)
- [Bug 3849](#):²⁸ Container descriptor is shared across containers in one JVM
- [Bug 3689](#):²⁹ Possible royalty / patent issue with BouncyCastle jar IDEA Algorithm

¹² http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3941

¹³ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4222

¹⁴ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4403

¹⁵ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4442

¹⁶ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=5008

¹⁷ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=5026

¹⁸ http://bugzilla.globus.org/globus/show_bug.cgi?id=2535

¹⁹ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2651

²⁰ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2743

²¹ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2207

²² http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2651

²³ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2743

²⁴ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2899

²⁵ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2900

²⁶ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2955

²⁷ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2955

²⁸ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3849

²⁹ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3689

- [Bug 3891](#).³⁰ Public credentials of client in peer subject
- [Bug 3965](#).³¹ Credential refresh problems
- [Bug 4021](#).³² globus-start-container -containerDesc not working
- [Bug 4136](#).³³ At least one of the headers used in dispatch was not secured error
- [Bug 4146](#).³⁴ setting default container security via environment
- [Bug 4507](#).³⁵ Problem with corrupted CRL
- [Bug 4535](#).³⁶ Client security descriptor does not allow for GSI Transport configuration
- [Bug 4584](#).³⁷ security descriptor uses operation field name instead of QName
- [Bug 4837](#).³⁸ Username/password not working.
- [Bug 4846](#).³⁹ Authorization framwork should preserve the order of attributes
- [Bug 4893](#).⁴⁰ Improve ParameterPIP test
- [Bug 5076](#).⁴¹ Authorization interface declares serializable, but impls are not
- [Bug 5544](#).⁴² Interceptor initializes twice
- [Bug 5608](#).⁴³ More details in security logging please
- [Bug 5756](#).⁴⁴ allow developer to bypass secure msg consistency check
- [Bug 5757](#).⁴⁵ allow developer to bypass sending cert chain in secure message

5. Performance reports

Secure access of WSRF and WSN operations using GSI Transport and GSI Secure Message have been measured. Test reports are in [here](#)⁴⁶.

³⁰ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3891

³¹ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3965

³² http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4021

³³ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4136

³⁴ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4146

³⁵ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4146

³⁶ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4535

³⁷ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4584

³⁸ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4837

³⁹ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4846

⁴⁰ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4893

⁴¹ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=5076

⁴² http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=5544

⁴³ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=5608

⁴⁴ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=5756

⁴⁵ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=5757

⁴⁶ [../common/javawscore/java_ws_core_wsrf_perf.html](http://common/javawscore/java_ws_core_wsrf_perf.html)

GT4 Java WS A&A Release Notes

Table of Contents

1. Component Overview	1
2. Feature Summary	1
3. Summary of Changes in Java WS A&A	3
4. Bug Fixes	4
5. Known Problems	5
6. Technology dependencies	6
7. Tested platforms	6
8. Backward compatibility summary	7
9. Associated Standards	7
10. For More Information	8

<titleabbrev>Release Notes</titleabbrev>

1. Component Overview

The Web Services portion of GT 4.2.0 uses SOAP over HTTP for communicating messages. WS Authentication & Authorization in Java (Java WS A&A) implements the WS-Security standard and the WS-SecureConversation specification to provide message protection for SOAP messages. Features include:

- authentication of the sender
- encryption of the message
- integrity protection of the message
- replay attack protection

Java WS A&A provides a secure channel by using HTTP over SSL/TLS (HTTPS) for transporting the messages. This security mechanism supports all of the security features provided by SSL/TLS with the addition of support for X.509 *Proxy Certificates*. The Authorization Framework component of Java WS A&A provides the infrastructure to process attributes and protect resource access based on access policy. It allows for authorization policy to be configured and enforced at various levels of granularity (container, service or resource). It also provides client-side authorization to allow clients to authorize the services they access. The framework is pluggable and can be configured to use custom mechanisms for attribute collection and policy evaluation. It also provides multiple authorization module implementations; for example, support for gridmap-based authorization, a callout module that uses the SAML protocol to query an external service for an authorization decision and such.

2. Feature Summary

2.1. Authentication/message protection features

Features new in GT 4.2.0

None.

Other Supported Features

- Compliance with published IBM/Microsoft WS-Trust and WS-SecureConversation specifications
- Compliance with the Web Services Security 1.0 standard
- HTTPS support
- Message encryption, integrity protection and replay attack prevention
- Establishment of a session key for light-weight message protection

Deprecated Features

- None.

2.2. Authorization features

Features new in GT 4.2.0:

- *Enhanced server-side attributed-based authorization framework:* The server-side authorization framework has been reworked to support attribute based authorization with delegation of rights. The framework allows for configuring a chain of Policy Information Points(PIPs) and Policy Decision Points(PDPs) and a combining algorithm that processes the individual decisions returned by the PDPs. Some of the key changes from the previous versions are:
 - Java Server side authorization framework has been moved to an independent module. Refer to Changes Summary for details.
 - Authorization framework uses a set of attributes to identify entities
 - The authorization engine uses Java Security provider framework to allow different combining algorithms to be plugged in.
 - A default implementation of permit override combining algorithm, which looks for a permit decision chain, to allow for fine grained delegation of rights.

Refer [Chapter 5, Architecture and design overview](#) for detailed information on the architecture.

- *Host or Self Authoriation:* Support for a pluggable PDP that does host authorization, and if that fails, tries self authorization.
- The security descriptor framework, used to configure security properties for the security framework has been enhanced. Detailed information about the framework is provided [Java WS A&A Security Descriptor Framework](#).

Other Supported Features

- Authorization based on `grid-mapfile` and other access control lists.
- Ability to implement custom authorization modules.
- A SAML callout authorization module enables outsourcing of authorization decisions to an authorization service (e.g. PERMIS).

Deprecated Features

- None

3. Summary of Changes in Java WS A&A

3.1. Summary of Changes in message/transport-level security

The following changes have occurred for Message/Transport-level Security since GT 4.0.x :

- Added support for signing policy enforcement. Disabling the enforcement is provided directly by the CoG JGlobus library, [Section 2, “Signing Policy Location”](#).
- The security descriptor framework, used to configure security properties for the security framework has been enhanced. Detailed information about the framework is provided at [Chapter 1, Security Descriptors Introduction](#).
- Java WS Authentication code honors environment variables to pick up credential to use as described [here](#)¹.
- Java WS Authentication code allows configuration of trust certificate in non-default location as described [here](#)².

3.2. Summary of Changes in WS Authorization Framework

The server side authorization framework has been reworked to support attribute-based authorization. The APIs and framework have been enhanced to deal with a representation where each entity is identified by a bag of attributes.

Also the default engine used for combining the individual Policy Decision Point(PDP) decision has been changed from a deny-override algorithm to a permit override scheme that looks for a chain of delegation of rights from the resource owner to the requestor.

Refer [Chapter 5, Architecture and design overview](#) for detailed information on the architecture.



Important

The WS authorization interfaces have been frozen as of the GT 4.1.2 release.



Note

All the PDPs that were distributed with the previous version have been ported to new framework and are supported.

3.2.1. Java WS Authorization Code reorganization *[post 4.1.3 only]*

The Java WS server side authorization code has been moved to a separate module called `authorization`. The work was tracked as part of [Bug 5559](#)³ and while this does not change any interface on the server side, it separates the code from the Java WS Core module.

A migration guide, that outlines the changes needed for services that build on Java WS Core is provided [here](#)⁴.

¹ http://bugzilla.mcs.anl.gov/globus/show_bug.cgi?id=4146

² http://bugzilla.mcs.anl.gov/globus/show_bug.cgi?id=3843

³ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=5559

⁴ http://dev.globus.org/wiki/Java_WS_Core/Independent_Java_Authz_Module

4. Bug Fixes

- [Bug 2535](http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2535):⁵ <proxy-file> causes container to fail
- [Bug 2651](http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2651):⁶ /dev/random vs. /dev/urandom
- [Bug 2743](http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2743):⁷ grid-mapfile location should be in global security descriptor
- [Bug 2207](http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2207):⁸ Missing security error 'timestampNotOk'
- [Bug 2651](http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2651):⁹ /dev/random vs. /dev/urandom
- [Bug 2743](http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2743):¹⁰ grid-mapfile location should be in global security descriptor
- [Bug 2899](http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2899):¹¹ relative path does not work for credentials in Security Descriptor
- [Bug 2900](http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2900):¹² Job submission does not work using relative path in global_security_descriptor.xml and absolute path in sudoers.
- [Bug 2955](http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2955):¹³ Job submission fails when container is started from non GLOBUS_LOCATION
- [Bug 2969](http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2969):¹⁴ Too relaxed rules on DN comparisons (all versions of GT)
- [Bug 3849](http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3849):¹⁵ Container descriptor is shared across containers in one JVM
- [Bug 3689](http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3689):¹⁶ Possible royalty / patent issue with BouncyCastle jar IDEA Algorithm
- [Bug 3891](http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3891):¹⁷ Public credentials of client in peer subject
- [Bug 3965](http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3965):¹⁸ Credential refresh problems
- [Bug 4021](http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4021):¹⁹ globus-start-container -containerDesc not working
- [Bug 4136](http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4136):²⁰ At least one of the headers used in dispatch was not secured error
- [Bug 4146](http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4146):²¹ setting default container security via environment
- [Bug 4507](http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4507):²² Problem with corrupted CRL

⁵ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2535

⁶ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2651

⁷ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2743

⁸ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2207

⁹ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2651

¹⁰ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2743

¹¹ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2899

¹² http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2900

¹³ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2955

¹⁴ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2955

¹⁵ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3849

¹⁶ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3689

¹⁷ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3891

¹⁸ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3965

¹⁹ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4021

²⁰ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4136

²¹ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4146

²² http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4146

- [Bug 4535](#)²³ Client security descriptor does not allow for GSI Transport configuration
- [Bug 4584](#)²⁴ security descriptor uses operation field name instead of QName
- [Bug 4837](#)²⁵ Username/password not working.
- [Bug 4846](#)²⁶ Authorization framework should preserve the order of attributes
- [Bug 4893](#)²⁷ Improve ParameterPIP test
- [Bug 5076](#)²⁸ Authorization interface declares serializable, but impls are not
- [Bug 5544](#)²⁹ Interceptor initializes twice
- [Bug 5608](#)³⁰ More details in security logging please
- [Bug 5756](#)³¹ allow developer to bypass secure msg consistency check
- [Bug 5757](#)³² allow developer to bypass sending cert chain in secure message

5. Known Problems

The following problems and limitations are known to exist for Java WS A&A at the time of the 4.2.0 release:

5.1. Limitations

- No known limitations exist.

5.2. Outstanding bugs

- [Bug 2362](#)³³ location of user proxy for java inconsistencies
- [Bug 2445](#)³⁴ Holder problem
- [Bug 2907](#)³⁵ Secure Conversation (Encryption) does not provide any message level security for the SOAP headers
- [Bug 3027](#)³⁶ Kerberos based authentication option for GT4
- [Bug 3171](#)³⁷ add RFC 2253 principal name to JAAS subject
- [Bug 3449](#)³⁸ ERROR container.GSIServiceThread

²³ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4535

²⁴ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4584

²⁵ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4837

²⁶ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4846

²⁷ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4893

²⁸ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=5076

²⁹ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=5544

³⁰ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=5608

³¹ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=5756

³² http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=5757

³³ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2362

³⁴ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2445

³⁵ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2907

³⁶ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3027

³⁷ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3171

³⁸ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3449

- [Bug 3603](#).³⁹ Remote exceptions thrown contain server specific information
- [Bug 3928](#).⁴⁰ IPv6 addresses in reverse lookups - fix or faq?
- [Bug 3941](#).⁴¹ Expired credentials detected - candidate for sec error msg improvements
- [Bug 4222](#).⁴² Allow for credential refresh in subscriptions
- [Bug 4403](#).⁴³ Secure calls for secure context establishment
- [Bug 4442](#).⁴⁴ Security descriptor refresh
- [Bug 5008](#).⁴⁵ voms-proxy-init creates non-critical KeyUsage extension which causes Java GSI to raise exception
- [Bug 5026](#).⁴⁶ Signarure validation failure on GRAM/RFT interaction on some cases

6. Technology dependencies

Java WS A&A depends on the following GT components:

- Java WS Core.

Authentication and message-protection depends on the following 3rd party software:

- Apache WSFX Security Libraries
- PureTLS Libraries
- BouncyCastle JCE provider
- Cryptix Libraries
- Apache XML Security Libraries

The authorization framework depends on the following 3rd party software:

- OpenSAML

7. Tested platforms

Java WS A&A should work on any platform that supports J2SE 1.3.1 or higher.

Tested Platforms for Java WS A&A:

- Linux (Red Hat 7.3)
- Windows 2000

³⁹ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3603

⁴⁰ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3928

⁴¹ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=3941

⁴² http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4222

⁴³ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4403

⁴⁴ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4442

⁴⁵ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=5008

⁴⁶ http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=5026

- Solaris 9

8. Backward compatibility summary

8.1. Authentication compatibility

Since GT 4.0.x release, some incompatible changes have been made:

- Security Descriptors: The security descriptor schema has changed since GT 4.0.x and the descriptors from GT 4.0.x cannot be used as is.
- Secure Conversation port type: The WS Addressing version in Java WS Core has been updated and the secure conversation port type has changed to reflect this. Therefore, GT 4.0.x secure conversation clients are incompatible with GT 4.2.x servers and vice versa.

8.2. Authorization compatibility

The authorization framework has been reworked as described in [Change Summary](#). The configuration and authorization interfaces have since changed and a [Migration Guide](#) is provided.

9. Associated Standards

Associated standards for Java WS A&A:

- [WS-Security](#)⁴⁷
- [WS-Security: X.509 Certificate Tokens](#)⁴⁸
- [WS-Security: Username Tokens](#)⁴⁹
- [WS-Trust](#)⁵⁰
- [WS-Secure Conversation](#)⁵¹
- [WS-I Basic Security Profile](#)⁵²
- [RFC 3820](#)⁵³ Proxy Certificates
- [RFC 2818](#)⁵⁴ HTTP over TLS
- [RFC 2246](#)⁵⁵ TLS
- [JAAS](#)⁵⁶

⁴⁷ <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>

⁴⁸ <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf>

⁴⁹ <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf>

⁵⁰ <ftp://www6.software.ibm.com/software/developer/library/ws-trust052004.pdf>

⁵¹ <ftp://www6.software.ibm.com/software/developer/library/ws-secureconversation052004.pdf>

⁵² <http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html>

⁵³ <http://www.faqs.org/rfcs/rfc3820.html>

⁵⁴ <http://www.faqs.org/rfcs/rfc2818.html>

⁵⁵ <http://www.faqs.org/rfcs/rfc2246.html>

⁵⁶ <http://java.sun.com/products/jaas/>

Associates standards for the authorization framework:

- [Simple Assertion Markup Language](#)⁵⁷
- [SAML Schema Protocol](#)⁵⁸

10. For More Information

See [Java WS A&A](#) for more information about this component.

DRAFT

⁵⁷ http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

⁵⁸ <http://www.oasis-open.org/committees/download.php/3407/oasis-sstc-saml-schema-protocol-1.1.xsd>