

## **GT 4.2.1 Java WS A&A Public Interfaces**

---

## **GT 4.2.1 Java WS A&A Public Interfaces**

---

---

# Table of Contents

1. APIs .....	1
1. Authorization Programming Model .....	1
2. Authentication/message protection Programming Model .....	1
3. API .....	1
2. Services and WSDL .....	3
1. Secure Conversation Service .....	3
2. SAML Authorization Callout .....	5
3. Framework-level Protocols .....	7
1. WS-Security .....	7
2. Transport (HTTPS) Security .....	7
4. Configuring client authentication and message/transport security .....	8
1. Interface introduction .....	8
2. Syntax of the interface .....	10
5. Authorization domain-level interface .....	15
1. Interface introduction .....	15
2. Syntax of the interface .....	15
6. PDP/PIP Links .....	20
7. Configuring .....	21
1. Configuring authorization .....	21
2. Configuring authentication/message protection .....	22
8. Environment variable interface .....	23
1. Environmental variables for WS Authentication & Authorization (Java) .....	23
A. Errors .....	24
Glossary .....	27

---

## List of Tables

4.1. Client side security properties .....	11
7.1. Configuring server side authentication and message/transport security .....	22
A.1. Java WS A&A Errors .....	25

---

# Chapter 1. APIs

Documentation for these interfaces can be found [here](#)<sup>1</sup>.

## 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 [Chapter 4, 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

---

<sup>1</sup> [http://www.globus.org/api/javadoc-4.2.1/globus\\_java\\_ws\\_core](http://www.globus.org/api/javadoc-4.2.1/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`

---

# Chapter 2. 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](http://www.ibm.com/developerworks/library/ws-trust/)<sup>1</sup> and [WS-SecureConversation](http://www-106.ibm.com/developerworks/library/ws-secon/)<sup>2</sup> 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>
```

---

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

<sup>2</sup> <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](#)<sup>3</sup>

---

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

- [WS-Trust XSD](#)<sup>4</sup>
- [WS-SecureConversation XSD](#)<sup>5</sup>
- 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 \(OGSA-AuthZ\)](#)<sup>6</sup> 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 \(SAML\)](#)<sup>7</sup> 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`

---

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

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

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

<sup>7</sup> [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=security](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`<sup>8</sup>
- `InvalidSetResourcePropertiesRequestContentFault`<sup>9</sup>
- `UnableToModifyResourcePropertyFault`<sup>10</sup>
- `InvalidResourcePropertyQNameFault`<sup>11</sup>
- `SetResourcePropertyRequestFailedFault`<sup>12</sup>
- `UnknownQueryExpressionDialectFault`<sup>13</sup>
- `InvalidQueryExpressionFault`<sup>14</sup>
- `QueryEvaluationErrorFault`<sup>15</sup>

## 2.5. WSDL and Schema Definition

- `OGSA-AuthZ Authorization Service WSDL`<sup>16</sup>

---

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

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

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

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

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

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

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

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

<sup>16</sup> [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](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 3. Framework-level Protocols

## 1. WS-Security

The framework implements the Web Services Security: SOAP Message Security<sup>1</sup>, Web Services Security: Username Token Profile<sup>2</sup> and Web Services Security: X.509 Token Profile<sup>3</sup> 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 (RFC3820<sup>4</sup>).

---

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

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

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

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

---

# Chapter 4. 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, 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();
}
```

## 2. Syntax of the interface

**Table 4.1. Client side security properties**

Number	Task	Stub Configuration	De- Co- tion
1.	Allows for configuration of credentials for authentication.	<p>Property:</p> <p><code>org.globus.axis.gsi.GSIConstants.GSI_CREDENTIALS</code></p> <p>Value equals the Instance of <code>org.ietf.jgss.GSSCredential</code>.</p>	Sec “C tial
2.	Allows for configuring client-side authorization.	<p>Property:</p> <p><code>org.globus.wsrsecurity.Constants.AUTHORIZATION</code></p> <p>Value equals the Instance of <code>org.globus.wsrsecurity.authorization.Authorization</code></p> <p>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.</p>	Rel Sec “A tion
3.	Enable GSI Secure Conversation with specified message protection level.	<p>1. Property:</p> <p><code>org.globus.wsrsecurity.Constants.GSI_SEC_CONV</code></p> <p>Values equal one of the following:</p> <ul style="list-style-type: none"> <li>• <code>Constants.ENCRYPTION</code></li> <li>• <code>Constants.SIGNATURE</code></li> </ul> <p>2. Property:</p> <p><code>org.globus.wsrsecurity.Constants.GSI_SEC_CONV_SECREPLY_UNNECESSARY</code></p> <p>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.</p> <p>3. Property:</p> <p>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.</p>	Rel tion Sec ver

Configuring client authentication and  
message/transport security

---

4.	<p>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>	<p>Property: <code>org.globus.axis.gsi.GSIConstants.GSI_MODE</code></p> <p>Value equals one of following:</p> <ol style="list-style-type: none"> <li>1. <code>GSIConstants.GSI_MODE_NO_DELEG</code>: No delegation is performed.</li> <li>2. <code>GSIConstants.GSI_MODE_LIMITED_DELEG</code>: Limited delegation is performed.</li> <li>3. <code>GSIConstants.GSI_MODE_FULL_DELEG</code>: Full delegation is performed.</li> </ol>	<p>Rel tion Sec ver</p>
5.	<p>Enables GSI Secure Transport with some protection level.</p>	<p>Property: <code>org.globus.gsi.GSIConstants.GSI_TRANSPORT</code></p> <p>Values equal one of the following:</p> <ul style="list-style-type: none"> <li>• <code>Constants.ENCRYPTION</code></li> <li>• <code>Constants.SIGNATURE</code></li> </ul>	<p>Rel tion Sec Tra</p>
6.	<p>Enables anonymous authentication. <i>This option only applies to GSI Secure Conversation and GSI Transport.</i></p>	<p>Property: <code>org.globus.wsrp.security.Constants.GSI_ANONYMOUS</code></p> <p>Value equals one of following:</p> <ol style="list-style-type: none"> <li>1. <code>Boolean.FALSE</code>: Anonymous authentication is disabled.</li> <li>2. <code>Boolean.TRUE</code>: Anonymous authentication is enabled.</li> </ol>	<p>Rel tion Sec ver and tion Sec Tra</p>

Configuring client authentication and  
message/transport security

---

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"> <li>• <code>Constants. ENCRYPTION</code></li> <li>• <code>Constants. SIGNATURE</code></li> </ul> <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>	Re tion Sec Me
8.	Enable WS-Security user-name/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>	Re tion "U nar wo

Configuring client authentication and  
message/transport security

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 Sec Me
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 "Tr 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 5. Authorization domain-level interface

## 1. Interface introduction

Configuration on the server side is done using [Chapter 1, 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](#)<sup>1</sup>

- **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:

---

<sup>1</sup> <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<sup>2</sup>](#).

## 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<sup>3</sup>](#). 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/>
```

---

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

<sup>3</sup> [../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 6. PDP/PIP Links

The following are links to PDPs and PIPs included in GT 4.2.1:

- [PDP Reference](#)
- [PIP Reference](#)

---

# Chapter 7. 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 ????
- Security descriptors programmatically. Refer to ????

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 ????, specifically [Section 3, “Authorization policy”](#).
- Security descriptors programmatically. Refer to ????
- 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 7.1. Configuring server side authentication and message/transport security**

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

---

# Chapter 8. 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.

---

# Appendix A. Errors

**Table A.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="802 296 1328 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> <li data-bbox="802 611 1328 800">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 <a href="#">Configuring Container Security Descriptor</a>.</li> <li data-bbox="802 827 1328 926">3. If you want to use host certificates, configure the container security descriptor as described <a href="#">Configuring Credentials</a>.</li> </ol>
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="802 959 1328 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.</li> </ol>
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="802 1194 1328 1398">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.</li> </ol>
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="802 1430 1328 1528">1. The container security descriptor should conform to the <a href="#">Container Security Descriptor Schema</a>.<sup>1</sup></li> <li data-bbox="802 1535 1328 1612">2. Refer to the "Caused by: " section for details on the specific element that is not correct.</li> </ol>

<sup>1</sup> [http://www.globus.org/toolkit/docs/4.2.1/security/container\\_security\\_descriptor.xsd](http://www.globus.org/toolkit/docs/4.2.1/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 1325 394">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 <a href="#">Java CoG Toolkit FAQ</a><sup>2</sup></li><li data-bbox="805 426 1325 520">2. On the server side, the trusted certificates can be configured as described in <a href="#">Trusted Certificates</a></li><li data-bbox="805 552 1325 646">3. On the client side, trusted certificates can be configured as described in <a href="#">Configuring Trusted Credentials</a></li></ol>

---

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

---

# Glossary

## P

public key

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