

GT 4.2.0 Java WS A&A Public Interfaces

DRAFT

GT 4.2.0 Java WS A&A Public Interfaces

DRAFT

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 Reference	20
1. Introduction	20
2. Access Control List PDP	20
3. GridMapAuthorization	21
4. Host Authorization	22
5. IdentityAuthorization	23
6. No Authorization	23
7. ResourceProperties Authorization	24
8. SAML Authorization Callout	25
9. SAML Authorization Assertion PDP	27
10. Self Authorization	27
11. Username Authorization	28
7. PIP Reference	32
1. Introduction	32
2. Container PIP	32
3. X509Bootstrap	34
4. SAML Authorization Assertion PIP	35
5. Parameter PIP	36
8. Configuring	38
1. Configuring authorization	38
2. Configuring authentication/message protection	39
9. Environment variable interface	40
1. Environmental variables for WS Authentication & Authorization (Java)	40
A. Errors	41
Glossary	44

List of Tables

4.1. Client side security properties	11
7.1. Attribute I	33
7.2. Attribute II	33
7.3. Attribute III	33
7.4. Attribute I	33
7.5. Attribute II	34
7.6. Attribute III	34
7.7. Attribute I	35
7.8. Attribute II	35
7.9. Attribute I	36
7.10. Attribute II	36
7.11. Attribute I	37
8.1. Configuring server side authentication and message/transport security	39
A.1. Java WS A&A Errors	42

DRAFT

Chapter 1. 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 [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

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 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](#)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 3. 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 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, 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 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- tion “C ing 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- tion “C ing atic ani
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"> • <code>Constants.ENCRYPTION</code> • <code>Constants.SIGNATURE</code> <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 “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>	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.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>	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 5. 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">  
    ...  
    <authz 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 6. 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.UsernameAuthzChainConfiguration">
        </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 7. 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 7.1. 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 7.2. 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 7.3. 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 7.4. 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 7.5. 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 7.6. 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 7.7. 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 7.8. 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 7.9. 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 7.10. 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 7.11. 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 8. 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 8.1. 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 9. 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

Appendix A. Errors

DRAFT

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="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 1614">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 1323 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 1323 520">2. On the server side, the trusted certificates can be configured as described in Trusted Certificates<li data-bbox="805 548 1323 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

P

public key

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

DRAFT