Java WS Core : Developer's Guide

Introduction

This guide contains information of interest to developers working with Java WS Core. It provides reference information for application developers, including APIs, architecture, procedures for using the APIs and code samples.
# Table of Contents

1. Before you begin .......................................................................................................................... 1
   1. Feature summary ...................................................................................................................... 1
   2. Tested platforms ...................................................................................................................... 1
   3. Backward compatibility summary ......................................................................................... 2
   4. Technology dependencies .................................................................................................... 3
   5. Security Considerations for Java WS Core ......................................................................... 4

2. Programming Recipes ..................................................................................................................... 5
   1. Service Scope and Lifetime ..................................................................................................... 5
   2. Operation Providers ................................................................................................................ 6
   3. Connecting to Services .......................................................................................................... 7
   4. Recipes about working with Endpoint Addressing ............................................................... 9
   5. Object Serialization .............................................................................................................. 12
   6. Notifications ......................................................................................................................... 13
   7. JNDI, Resources and Objects .............................................................................................. 19
   8. Asynchronous Work and Threads ....................................................................................... 21
   9. Querying Properties .............................................................................................................. 23
   10. Build Recipes ..................................................................................................................... 27
   11. Controlling client connection settings ............................................................................. 33
   12. Writing and running tests .................................................................................................. 34
   13. Miscellaneous Recipes ....................................................................................................... 36

3. Tutorials and Presentations .......................................................................................................... 40
   1. Tutorials ............................................................................................................................... 40
   2. Presentations ........................................................................................................................ 40

4. Samples ...................................................................................................................................... 41
   1. Counter Sample ..................................................................................................................... 41
   2. Management Sample ............................................................................................................. 42

5. Architecture and design overview .............................................................................................. 44

6. APIs ........................................................................................................................................ 45
   1. Programming Model Overview .......................................................................................... 45
   2. Component API .................................................................................................................... 45

7. Services and WSDL ................................................................................................................... 48

I. Java WS Core Commands ........................................................................................................ 5
   globus-start-container ........................................................................................................... 50
   globus-stop-container ......................................................................................................... 51
   globus-start-container-detached ......................................................................................... 53
   globus-stop-container-detached .......................................................................................... 54
   wsrfsr-destroy ...................................................................................................................... 55
   wsrfsr-set-termination-time ............................................................................................... 57
   wsrfsr-query ........................................................................................................................... 59
   wsrfsr-get-property ............................................................................................................. 61
   wsrfsr-get-properties .......................................................................................................... 63
   wsrfsr-insert-property ......................................................................................................... 65
   wsrfsr-update-property ........................................................................................................ 67
   wsrfsr-delete-property ......................................................................................................... 69
   wsn-get-current-message .................................................................................................... 71
   wsn-pause-subscription ...................................................................................................... 73
   wsn-resume-subscription .................................................................................................... 75
   wsn-subscribe ....................................................................................................................... 77
   globus-deploy-gar .............................................................................................................. 79
   globus-undeploy-gar ............................................................................................................ 81
   globus-check-environment ................................................................................................. 83
List of Tables

2.1. Service Scopes ................................................................. 6
2.2. Evaluator interfaces ......................................................... 25
3. Options .............................................................................. 50
4. Common options ............................................................... 52
5. Shutdown options .............................................................. 52
6. Options .............................................................................. 53
7. Options .............................................................................. 54
8. Common options ............................................................... 56
9. Command-specific options ............................................... 57
10. Common options ............................................................. 58
11. Common options ............................................................. 60
12. Common options ............................................................. 62
13. Common options ............................................................. 64
14. Common options ............................................................. 66
15. Common options ............................................................. 68
16. Common options ............................................................. 70
17. Common options ............................................................. 72
18. Common options ............................................................. 74
19. Common options ............................................................. 76
20. Command-specific options ............................................... 77
21. Common options ............................................................. 78
22. Options .............................................................................. 79
23. Supported property-value pairs ......................................... 79
24. Options .............................................................................. 81
25. Options .............................................................................. 84
26. Options .............................................................................. 85
27. Options .............................................................................. 86
28. Common options ............................................................. 88
29. Common options ............................................................. 90
30. Command-specific options ............................................... 91
31. Common options ............................................................. 92
32. Common options ............................................................. 94
33. Command-specific options ............................................... 95
34. Common options ............................................................. 96
35. Common options ............................................................. 98
36. Command-specific options ............................................... 99
37. Common options ............................................................. 100
38. Common options ............................................................. 102
8.1. General configuration parameters ..................................... 103
8.2. Standalone/embedded container-specific configuration parameters ............................................. 104
8.3. Container Thread Parameters .......................................... 105
8.4. Default container thread pool settings ................................ 105
8.5. Axis Standard Parameters .............................................. 107
8.6. Java WS Core Parameters .............................................. 107
8.7. ResourceHomeImpl parameters ...................................... 109
9.1. Globus standard environment variables ............................ 112
9.2. Launch script specific environment variables ................... 112
9.3. Options supported by the GLOBUS_OPTIONS environment property ............................................ 113
11.1. Java WS Core Errors .................................................... 118
Chapter 1. Before you begin

1. Feature summary

New Features in the GT 4.2 release

- Persistent HTTP/S connection support (client and server side)
- Support for SOAP with Attachments (DIME, MIME, and MTOM formats). See the SOAP Attachments section for details.
- Dynamic deployment support for the standalone container. See the Java WS Core Container Dynamic Deploy Design Document for more information.
- WS-Enumeration\(^1\) support. See the Java WS Core WS-Enumeration Design Document for more information.
- JBOSS 4.0.x support.
- The TargetedXPath query dialect implementation that enables simpler and more efficient querying of resource property documents.

Other Supported Features

- A standalone and embeddable container
- Tomcat 4.1, 5.0, and 5.5 support
- Basic API for resource persistence and recovery
- Persistent subscriptions support
- Automatic service and ResourceHome activation on startup
- Operation providers

Deprecated Features

- None

2. Tested platforms

Java WS Core should work on any platform that supports J2SE 1.4.1 or higher.

Tested platforms for Java WS Core:

- Linux (Debian 3.1)
- Windows XP
- Solaris 9

Tested JVMs for Java WS Core:

\(^1\) http://www.w3.org/Submission/WS-Enumeration/
• Sun JVM\(^2\) 1.4.2, 1.5.0, and 1.6.0 RC
• IBM JVM\(^3\) 1.4.1, and 1.4.2
• BEA JRockit JVM\(^4\) 1.5.0

JVM notes:
• GCJ\(^5\) is not supported.
• If using IBM JVM 1.4.1 please see bug 2828\(^6\) for more information.

Tested containers for Java WS Core:
• Java WS Core container
• Tomcat 4.1.34
• Tomcat 5.0.30
• Tomcat 5.5.20
• JBoss 4.0.2

3. Backward compatibility summary

Protocol changes since GT version 4.0.x:
• Upgrade to WS-Addressing 1.0
• Upgrade to WSRF 1.2
• For an overview of the changes and links to the new specifications, please see this document\(^7\).
• For a migration guide for GT 4.0.x services, please see this document\(^8\).

API changes since GT version 4.0.x:
• Java WS Core now requires Java 1.5 to compile and run. Java 5 provides new concurrency capabilities as well as Annotations and Generics (as well as some other features). For details on the new features, please see this overview\(^9\).
• While it is possible that Java WS Core could run under Retroweaver\(^10\) or Retrotranslator\(^11\), this has not been tested. Java 5 is available for most platforms from Sun http://java.sun.com/javase/downloads/index_jdk5.jsp
• The faultCause field in the BaseFaultType\(^12\) class is no longer an Object. Instead it has been changed to a BaseFaultTypeFaultCause. This class, in turn, contains an object that represents the actual fault cause called causeElement. This change was made in order to support interoperability with other groups.

\(^2\) http://java.sun.com/j2se/
\(^3\) http://www-106.ibm.com/developerworks/java/jdk/
\(^4\) http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/jrockit
\(^5\) http://gcc.gnu.org/java/
\(^6\) http://bugzilla.globus.org/globus/show_bug.cgi?id=2828#c4
\(^7\) http://dev.globus.org/images/d/6b/CoreSpecificationUpgradeVer11.pdf
\(^8\) http://dev.globus.org/wiki/Java_WS_Core/Final_WSRF_Migration_Guide
\(^9\) http://java.sun.com/j2se/1.5.0/docs/relnotes/features.html
\(^10\) http://retroweaver.sourceforge.net/
\(^11\) http://retrotranslator.sourceforge.net/
\(^12\) http://docs
Schema changes since GT version 4.0.x:

- Please see the Protocol Changes section

Other changes since GT version 4.0.x:

- Java 1.5 is required to use Java WS Core 4.2.

### 4. Technology dependencies

Java WS Core depends on the following GT components:

- **Java CoG Kit**[^13]

Java WS Core depends on the following 3rd party software:

- **Apache Xerces**[^14]
- **Apache XML Security**[^15]
- **Apache Axis**[^16]
- **Apache Xalan**[^17]
- **Apache Addressing**[^18]
- **Apache Commons BeanUtils**[^19]
- **Apache Commons CLI**[^20]
- **Apache Commons Collections**[^21]
- **Apache Commons Digester**[^22]
- **Apache Commons HttpClient**[^23]
- **Concurrent Library**[^24]
- **Apache Directory (Naming)**[^25]

Please see [Java WS Core Technical Dependencies Details](#) for details.

[^13]: http://www.globus.org/cog/java/
[^14]: http://xml.apache.org/xerces2-j/
[^15]: http://xml.apache.org/security/
[^16]: http://ws.apache.org/axis/
[^17]: http://xml.apache.org/xalan-j/
[^18]: http://ws.apache.org/ws-fx/addressing/
[^19]: http://jakarta.apache.org/commons/beanutils/
[^20]: http://jakarta.apache.org/commons/cli/
[^21]: http://jakarta.apache.org/commons/collections/
[^22]: http://jakarta.apache.org/commons/digester/
[^23]: http://jakarta.apache.org/commons/httpclient/
[^25]: http://directory.apache.org/subprojects/naming/
5. Security Considerations for Java WS Core

5.1. Permissions of service configuration files

The service configuration files such as `jndi-config.xml` or `server-config.wsdd` (located under `$GLOBUS_LOCATION/etc/<gar>/` directory) may contain private information such as database passwords, etc. Ensure that these configuration files are only readable by the user that is running the container. The deployment process automatically sets the permissions of the `jndi-config.xml` and `server-config.wsdd` files as user readable only. However, this might not work correctly on all platforms and this does not apply to any other configuration files.

5.2. Permissions of persistent data

The services using subscription persistence API or other basic persistence helper API will store all or part of its persistent data under the `~/.globus/persisted` directory. Ensure that the entire `~/.globus/persisted` directory is only readable by the user running the container.

5.3. Invocation of non-public service functions

A client can potentially invoke a service function that is not formally defined in the WSDL but it is defined in the service implementation class. There are two ways to prevent this from happening:

1. Define all service methods in your service class as either `private` or `protected`.

2. Configure appropriate `allowedMethods` or `allowedMethodsClass` parameter in the service deployment descriptor (please see Configuring Java WS Core for details).
Chapter 2. Programming Recipes

1. Service Scope and Lifetime

1.1. How can I force my service to initialize when the container starts?

By default, a service is initialized on the first invocation of a method in that service. However, there are frequently times when a service needs to run some time-consuming, global initialization. In this situation, it is better to have the service initialize when the container starts, so as to not impact the performance of the first method invocation. To have the service initialize on container startup, add the following to your deploy-service.wsdd descriptor:

```
<parameter name="loadOnStartup" value="true"/>
```

1.2. How can I have Service methods execute at lifetime (Startup/Shutdown) events?

If the service implements the `javax.xml.rpc.server.ServiceLifecycle` interface, the lifecycle methods will be called according to the `scope` setting as a service instance is created and destroyed.

For example, in `Application` scope, `destroy()` will be called on container shutdown, and in `Request` scope it will be called after the service method is called. (See next question)

1.3. How can I have ResourceHome (or any JNDI resource) methods execute at lifetime events?

A `ResourceHome` will be activated either on the first service invocation or, if "loadOnStartup" parameter is set to "true", during container startup. Both mechanisms trigger actual activation by looking up the ResourceHome in the JNDI directory. This initial lookup causes a proper MessageContext and/or JAAS subject to be associated with the current thread, instantiation of the object implementing the ResourceHome and, if the ResourceHome implements the `org.globus.wsr.f.jndi.Initializable` interface, the invocation of the `initialize()` function.

In fact, the same steps are performed upon initial lookup of any JNDI resource entry that uses the `org.globus.wsr.f.jndi.BeanFactory` class for its factory and is defined directly under a `service` entry in a `jndi-config.xml` file.

1.4. How do I set the scope of my service?

The `scope` setting of the service dictates when and how service instances are created. It is generally recommended that services use the Application scope (even though it is not currently the default) because it associates the security credentials with the thread. To set the scope, you add the following parameters to your deploy-server.wsdd:

---

1 [http://java.sun.com/j2ee/1.4/docs/api/javax/xml/rpc/server/ServiceLifecycle.html](http://java.sun.com/j2ee/1.4/docs/api/javax/xml/rpc/server/ServiceLifecycle.html)
This will specify to use the globus supplied RPCProvider and to set the scope to Application. This is the recommended setting for most services. If you have any question about what to use, set this.

There are two scope options:

**Table 2.1. Service Scopes**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Service/provider instances are created either on first invocation or on container startup. The behavior is determined by the value of the &quot;loadOnStartup&quot; parameter. This will work in the same way in both the stand-alone container and in Tomcat. If the service or the container is configured with a security descriptor, the appropriate credentials will be associated with the thread during activation (using JAAS). Also, during activation a basic Axis MessageContext will be associated with the thread with only Constants.MC_HOME_DIR, Constants.MC_CONFIGPATH, and the right target service properties set (see &quot;Obtaining standard MessageContext properties&quot; for details). If service or providers implement the javax.xml.rpc.server.ServiceLifecycle interface, the lifecycle methods will be called accordingly.</td>
</tr>
<tr>
<td>Request</td>
<td>One instance is created per invocation. This is the default (if scope parameter is not set in the deployment descriptor). Service/Provider instances are created per invocation, ServiceLifecycle methods called right before and after service method invocation, no JAAS credentials during ServiceLifecycle methods.</td>
</tr>
</tbody>
</table>

2. **Operation Providers**

2.1. **How do I use Operation Providers?**

GT3 introduced the concept of *operation providers* where a service could be composed of different parts/classes. Java WS Core also supports this functionality. In GT3 operation providers had to implement a specific interface. In Java WS Core *no* such interface is required. In fact, an operation provider is not in any way different from a standard web service. That means that *any* web service implementation can automatically be used as an operation provider (as long as it uses common or standard interfaces to operate on resources).

To enable operation provider support for your service, make the following changes to the service deployment descriptor:

1. Change the value of the `provider` attribute to `Handler`.
2. Add a `handlerClass` parameter with a value of `org.globus.axis.providers.RPCProvider`.
3. Specify providers in the `providers` parameter.
   The value of the parameter is a space-separated list of either provider names or class names.
4. Add or change the value of the `scope` parameter to `Application` or `Request`.
The following is an example of a modified service deployment descriptor:

```xml
<service name="SubscriptionManagerService" provider="Handler"
    use="literal" style="document">
    <parameter name="allowedMethods" value="*"/>
    <parameter name="scope" value="Application"/>
    <parameter name="providers" value="GetRPProvider org.globus.wsrf.impl.lifetime.SetTerminationTimeProvider PauseSubscriptionProvider"/>
    <parameter name="handlerClass" value="org.globus.axis.providers.RPCProvider"/>
    <parameter name="className" value="org.globus.wsrf.impl.notification.ResumeSubscriptionProvider"/>
    <wsdlFile>share/schema/core/notification/subscription_manager_service.wsdl</wsdlFile>
</service>
```

**Note**

The operations defined in the `className` service always overwrite the providers' operations. That is, if one provider defines the same method as the service specified in the `className` parameter, the operation will be invoked on the service. Also, if two providers define the same method, the first one specified in the `providers` parameter will be invoked.

### 2.2. How can I expose my own class as an Operation Provider for other services to use?

Any service can provide an object for other service to include as a operation provider. The consuming service simply needs to add the fully qualified classname (FQCN) to the list of providers. However, if you wish to allow users to refer to your class by a friendly name instead of FQCN, you need to add an entry to the container's main deployment file element of the `main` deployment descriptor (`etc\globus_wsrf_core\server-config.wsdd`).

For example:

```xml
<globalConfiguration>
...
    <parameter name="GetRPProvider" value="org.globus.wsrf.impl.properties.GetResourcePropertyProvider"/>
...
</globalConfiguration>
```

### 3. Connecting to Services

These recipes are all about connecting to services running in a Java WS Core container.
3.1. How do I connect to a service via WSRF?

There are several ways to connect to a service. The simplest is to use the AddressingLocator that is created as part of the WSDL2Java execution. The AddressingLocator class can be used to get a stub for a service by passing the Apache Addressing EndpointReferenceType parameter. Each service has its own AddressingLocator which can be used to acquire a port for the remote service. The port can then be used to access the service. If an EndpointReferenceType is returned from a method on that service, it can be used to acquire the port from the AddressingLocator and will be associated with the resource context created by the service.

```java
//Specify the service address
String url = "http://localhost:8080/wsrf/services/Version";
//Create the EndpointReferenceType
EndpointReferenceType epr = new EndpointReferenceType();
epr.setAddress(new Address(url));
//Create the AddressingLocator for the service
VersionServiceAddressingLocator locator = new VersionServiceAddressingLocator();
//Get the port for the service
VersionServicePortType port = locator.getVersionPort(epr);
port.getVersion();
```

3.2. How do I connect to a service in the same Java Virtual Machine in memory?

Services in the container can be invoked locally. Local invocations work just like remote invocations (all handlers are called, messages get serialized/deserialized) but messages do not travel over the network - everything happens in memory.

Local invocations can only be made on the server side. URLs with "local" protocol name are used for local invocations.

To invoke a service locally, do the following:

1. Create a service URL with "local" protocol:

   ```java
   URL url = new URL("local:///wsrf/services/MyService");
   ```

2. And as normal make the call:

   ```java
   MyServiceAddressingLocator locator = new MyServiceAddressingLocator();
   MyService port = locator.getServicePort(url);
   port.hello();
   ```

That's all. By default the local invocations are made using the default instance of the AxisServer engine. It is possible to make local invocations using a different AxisServer engine instance if there is a MessageContext associated with the current thread (the MessageContext should have a reference to the desired AxisServer engine instance). To make a local invocation using non-default AxisServer engine add the following (using above example):
import org.globus.axis.transport.local.LocalTransportUtils;
...

MyService port = ...  
LocalTransportUtils.enableLocalTransport((Stub)port);  
port.hello();

The "local" protocol URL handler is automatically registered by Java WS Core. However, sometimes it might be necessary to install the handler explicitly. To register the "local" URL protocol handler do the following:

import org.globus.axis.transport.local.LocalTransportUtils;
...
LocalTransportUtils.init();  
...

This step must be done before creating URLs with "local" protocol. Also, make sure that axis-local.jar is accessible from the system classloader.

4. Recipes about working with Endpoint Addressing

4.1. How can my service object obtain information about its Endpoint Address?

In most cases, a service will need to return the endpoint information of the container to a client. Unfortunately, getting that information might not be easy. The only reliable way of getting the container endpoint information is to extract it from the MessageContext.TRANS_URL property of the MessageContext/ResourceContext associated with the current thread.

To obtain base container endpoint information use the ServiceHost\(^2\) API. For example:

import org.globus.wsrf.container.ServiceHost;
...

URL containerBaseUrl = ServiceHost.getBaseURL();
...

The above will return the base container URL such as http://localhost:8080/wsrf/services/.

To obtain service endpoint information use the ResourceContext\(^3\) API. For example:

---

\(^2\) http://www.globus.org/api/javadoc-4.2.0/org/globus/wsrf/container/ServiceHost.html  
\(^3\) http://www.globus.org/api/javadoc-4.2.0/org/globus/wsrf/ResourceContext.html
import org.globus.wsrf.ResourceContext; ... URL serviceUrl = ResourceContext.getResourceContext().getServiceURL(); 

The above will return the service URL such as http://localhost:8080/wsrf/services/MyService.

To obtain WS-Addressing endpoint for the service use the AddressingUtils\(^4\) API. For example:

```java
import org.apache.axis.message.addressing.EndpointReferenceType;
import org.globus.wsrf.utils.AddressingUtils;
...
EndpointReferenceType containerEndpoint = AddressingUtils.createEndpointReference(null);
...
```

The above will create a EndpointReferenceType object initialized with the Address field set to the service URL (as before) and empty reference properties. Also, you can pass a non-null ResourceKey instance to the createEndpointReference() function to create an endpoint for a specific resource. The reference properties field of the created EndpointReferenceType object will be set to the given ResourceKey.

**Note**

The ServiceHost API will return the correct information and AddressingUtils API will work correctly only if called from the same thread as the service method was invoked from.

4.2. How do I use WS-Addressing with low level API's such as javax.xml.rpc.Call?

If you are using the javax.xml.rpc.Call object directly, you can pass the addressing information by setting a Constants.ENV_ADDRESSING_REQUEST_HEADERS property on the call object.

For example:

```java
Service service = new Service();
Call call = (Call) service.createCall();
String url = "http://localhost:8080/axis/services/Version";
AddressingHeaders headers = new AddressingHeaders();
headers.setTo(new To(url));
// pass the addressing info to the addressing handler
call.setProperty(Constants.ENV_ADDRESSING_REQUEST_HEADERS, headers);
call.setTargetEndpointAddress(new URL(url));
call.setOperationName(new QName(url, "getVersion")); // url here is just a namespace
String ret = (String) call.invoke(new Object[]);
```

\(^4\)http://www.globus.org/api/javadoc-4.2.0/org/globus/wsrf/utils/AddressingUtils.html
4.3. How do I use ReferenceParameters to specify resource identity for my service?

In the WS-RF and WS-N specifications, the WS-Addressing ReferenceParameters are used to carry resource identity information. In previous versions of the WSRF/WSN specification, ReferenceProperties was used for the same purpose. The resource identity can be anything as long as it serializes as a XML element. The ReferenceParameters are serialized as separate SOAP headers in the SOAP envelope.

The Apache Addressing library only allows a DOM Element or a SOAPElement to be a reference parameters.

For example, create ReferenceParametersType and fill it with resource key info:

```java
// create a reference property
QName keyName = new QName("http://axis.org", "VersionKey");
String keyValue = "123";
SimpleResourceKey key = new SimpleResourceKey(keyName, keyValue);
ReferenceParametersType props = new ReferenceParametersType();
// convert to SOAPElement and add to the list
props.add(key.toSOAPElement());
...
```

Then pass it to AddressingHeaders:

```java
...

Service service = new Service();
Call call = (Call) service.createCall();
String url = "http://localhost:8080/axis/services/Version";
AddressingHeaders headers = new AddressingHeaders(); headers.setTo(new To(url));
headers.setReferenceParameters(props); // pass the addressinginfo to the addressing handler
call.setProperty(Constants.ENV_ADDRESSING_REQUEST_HEADERS, headers);
call.setTargetEndpointAddress(new URL(url));
call.setOperationName(new QName(url, "getVersion")); // url here is just a namespace
String ret = (String) call.invoke(new Object[]);
```

Or set it on EndpointReferenceType:

```java
...

String url = "http://localhost:8080/axis/services/Version";
EndpointReferenceType epr = new EndpointReferenceType();
epr.setAddress(new Address(url));
epr.setParameters(props);
VersionServiceAddressingLocator locator = new VersionServiceAddressingLocator();
VerionServicePortType port = locator.getVersionPort(epr);
port.getVersion();
```
5. Object Serialization

5.1. How can I handle information defined as xsd:any in my service?

Java Beans generated by Apache Axis that represent a XML type with the xsd:any content implement the org.apache.axis.encoding.AnyContentType interface and have get_any() and set_any() methods. There are several API to help you deal with the AnyContentType content.

To convert AnyContentType content to a Java object use the ObjectDeserializer API. For example:

```java
AnyContentType bean = ...;
Integer value = (Integer) ObjectDeserializer.getFirstAsObject( bean), Integer.class);
```

To convert a Java object into a type that can be used with AnyContentType content use the ObjectSerializer API. For example:

```java
// convert Java object into SOAPElement EndpointReferenceType object = ...;
QName elementName = new QName("http://example.com", "EPR");
SOAPElement element = ObjectSerializer.toSOAPElement(object, elementName);
// set the SOAPlement as Any content
AnyContentType bean = ...;
AnyHelper.setAny(bean, element);
```

To examine the raw AnyContentType content use the AnyHelper API to serialize it as XML. For example:

```java
AnyContentType bean = ...;
String contents = AnyHelper.toSingleString(bean);
System.out.println("Contents: " + contents);
```

5.2. How can I serialize my Java Object to a file?

You can use the ObjectSerializer API to serialize the Java object into a file (in XML format):

```java
// object to serialize EndpointReferenceType epr = ...;
// root element name
QName elementName = new QName("http://example.com", "EPR");
FileWriter out = null;
try {
    out = new FileWriter("epr.xml");
    ObjectSerializer.serialize(out, epr, elementName);
```

5 http://www.globus.org/api/javadoc-4.2.0/org/globus/wsrf/encoding/ObjectDeserializer.html
6 http://www.globus.org/api/javadoc-4.2.0/org/globus/wsrf/encoding/ObjectSerializer.html
7 http://www.globus.org/api/javadoc-4.2.0/org/globus/wsrf/utils/AnyHelper.html
8 http://www.globus.org/api/javadoc-4.2.0/org/globus/wsrf/encoding/ObjectSerializer.html
Similarly, use can use the ObjectDeserializer\(^9\) API to deserialize a file containing XML data into a Java object:

```java
// deserialized object EndpointReferenceType epr = ...; FileInputStream in = null; try { in = new FileInputStream("epr.xml"); epr = (EndpointReferenceType)ObjectDeserializer.deserialize( new InputSource(in), EndpointReferenceType.class); } catch (IOException e) { System.err.println("Error: " + e.getMessage()); } finally { if (in != null) { try { in.close(); } catch (Exception ee) {} } }
```

### 6. Notifications

6.1. How can I set up my client/service to receive notifications

There are a few steps involved in setting up and receiving notifications:

#### 6.1.1. Step 1: Implement the callback

The notification consumer application must provide an implementation of the NotifyCallback\(^10\) interface. The deliver function of the interface will be invoked whenever a notification for that consumer arrives.

Note

The deliver function should be thread-safe as multiple notifications might come at once. Notifications might also come unordered and some might even be lost (due to network failures).

#### 6.1.1.1. Resource property notifications

For resource property notifications the message parameter will usually be of ResourcePropertyValueChangeNotificationElementType type. From that type you can retrieve the real notification message which contains the new and optionally the old value of the resource property. Example:

```java
import org.globus.wsrf.encoding.DeserializationException;
import org.globus.wsrf.utils.NotificationUtil;
import org.oasis.wsrf.properties.ResourcePropertyValueChangeNotificationType;
```

\(^9\) [http://www.globus.org/api/javadoc-4.2.0/org/globus/wsrf/encoding/ObjectDeserializer.html](http://www.globus.org/api/javadoc-4.2.0/org/globus/wsrf/encoding/ObjectDeserializer.html)

// Notification callback
public void deliver(List topicPath, EndpointReferenceType producer, Object message) {
    ResourcePropertyValueChangeNotificationType changeMessage = null;
    try {
        changeMessage = NotificationUtil.getRPValueChangeNotification(message);
    } catch (DeserializationException e) {
        // handle exception
    }
    Integer newValue = (Integer)ObjectDeserializer.getFirstAsObject(changeMessage.getNewValue(), Integer.class);
    System.out.println("New value: "+ newValue);
    if(changeMessage.getOldValue() != null) {
        Integer oldValue = (Integer)ObjectDeserializer.getFirstAsObject(changeMessage.getNewValue(), Integer.class);
        System.out.println("Old value: " + oldValue);
    }
}

The resource property values are of AnyContentType type. Please see the Section 5.1, “How can I handle information defined as xsd:any in my service?” section for more information on how to deal with such types.

6.1.1.2. Other notifications

For other non-resource property notifications the message parameter will either be of the type into which the message type maps into (if there is an appropriate type mapping defined) or of org.w3c.dom.Element type if there is no appropriate type mapping defined. Example:

public void deliver(List topicPath, EndpointReferenceType producer, Object message) {
    EndpointReferenceType epr = null;
    if (message instanceof Element) {
        // type mapping not defined, try to deserialize into right Java
        // type using ObjectDeserializer API.
        epr = (EndpointReferenceType)ObjectDeserializer.toObject((Element)message, EndpointReferenceType.class);
    } else if (message instanceof EndpointReferenceType) {
        // type mapping defined
        epr = (EndpointReferenceType)message;
    } else {
        // some other type
    }
}

The custom notification message type mappings can be defined in a client-server-config.wsdd file. This file can be deployed with your service (it must be placed directly under the etc/ directory in the GAR file). Please see the $GLOBUS_LOCATION/etc/globus_wsrf_core/client-server-config.wsdd file for an example. If your callback implementation will be used on the server-side, you might also need to define the type mappings in your server-config.wsdd.

6.1.2. Step 2: Start NotificationConsumerManager

In order to facilitate the receipt of notifications, start a NotificationConsumerManager by doing the following:
import org.globus.wsrf.NotificationConsumerManager;

...

NotificationConsumerManager consumer = null;

try {
    consumer = NotificationConsumerManager.getInstance();
    consumer.startListening();
    ...
} catch (...) {
    ...
}

Important

On the client when the consumer.startListening() is called an embedded container is actually started in the background. That embedded container is the same as the standalone container but configured with only one or two services needed to handle the notifications. Therefore, any client using notification consumer API will have the same dependencies on the libraries and configurations files as the basic standalone container code. Also, please check the Failed to acquire notification consumer home instance from registry error in Table 11.1, “Java WS Core Errors” if the consumer.startListening() call failed on the client.

On the server when the consumer.startListening() is called the container in which the service is running in is used to receive the notifications. Therefore, there are no extra dependencies.

6.1.3. Step 3: Register the callback

Register the callback implementation with the NotificationConsumerManager (once it is started) using the createNotificationConsumer function.

The createNotificationConsumer function returns an endpoint for this notification consumer.

Example:

import org.globus.wsrf.NotifyCallback;
import org.apache.axis.message.addressing.EndpointReferenceType;

...

MyCallback callback = new MyCallback();
EndpointReferenceType consumerEPR = consumer.createNotificationConsumer(callback);
...

class MyCallback implements NotifyCallback {
    ....
}
6.1.4. Step 4: Subscribe to the callback

Pass the endpoint returned by the `createNotificationConsumer` function to the subscribe call.

Example:

```java
import org.oasis.wsn.TopicExpressionType;
import org.oasis.wsn.Subscribe;
import org.oasis.wsn.SubscribeResponse;
import org.oasis.globus.wsrf.WSNConstants;
import org.globus.wsrf.WSRFConstants;

...

Subscribe request = new Subscribe();
request.setConsumerReference(consumerEPR);
TopicExpressionType topicExpression = new TopicExpressionType();
topicExpression.setDialect(WSNConstants.SIMPLE_TOPIC_DIALECT);
topicExpression.setValue(Counter.VALUE);
MessageElement element = (MessageElement)ObjectSerializer.toSOAPElement(topicExpression, WSNConstants.TOPIC_EXPRESSION);
FilterType filter = new FilterType();
filter.set_any(new MessageElement[] { element });
request.setFilter(filter);
port.subscribe(request);
...
```

6.1.5. Step 5: Clean up

Once done with the notifications, do the following clean up tasks.

**Step 5a: Destroy subscriptions resource.** Make sure to explicitly destroy the subscription resource or set its termination time. Example:

```java
import org.globus.wsrf.core.notification.SubscriptionManager;
import org.globus.wsrf.core.notification.service.SubscriptionManagerServiceAddressingLocator;
import org.oasis.wsrf.lifetime.Destroy;

...

SubscriptionManagerServiceAddressingLocator sLocator = new SubscriptionManagerServiceAddressingLocator();
SubscriptionManager manager = sLocator.getSubscriptionManagerPort( subResponse.getSubscriptionReference());
manager.destroy( new Destroy());
...
```

**Step 5b: Un-register the callback.** Make sure to call (especially in error cases) the `NotificationConsumerManager.removeNotificationConsumer()` function to unregister the callback from the NotificationConsumerManager.
Step 5c: Release resources. In addition, make sure to always call the `NotificationConsumerManager.stopListening()` function when finished using the `NotificationConsumerManager`. Otherwise, some resources might not be released. Example:

```java
... 
} catch(Exception e) {
...
}
finally {
    if (consumer != null) {
        try {
            consumer.stopListening();
        } catch (Exception ee) {
            //Handle Exception
        }
    }
}
```

6.2. How can I use a notification topic that is not tied to a ResourceProperty?

While the typical usage of WS-Notification within WSRF is to notify subscribers when a ResourceProperty changes, this is not a requirement. In fact, it is easy to set up a WS-Notification topic to support general messages. The Resource which will handle your topic should implement `org.globus.wsrf.TopicListAccessor`. This means your Resource needs to implement the `TopicList getTopicList()` method. The simplest way to do this, is to create an instance of `SimpleTopicList` in your resource like such:

```java
public class MyResource implements ResourceProperties, TopicListAccessor{
    private TopicList topicList;
    private ResourcePropertySet propertySet;
    private QName qname = new QName("http://www.globus.org/MyService", "MyResource");
    public MyResource(){
        topicList = new SimpleTopicList(this);
        propertySet = new ResourcePropertySet(qname);
    }
    public TopicList getTopicList(){
        return topicList;
    }
    public ResourcePropertySet getResourcePropertySet(){
        return propertySet;
    }
}
```

Once you have this, your resource is ready to work with topics. To create a new `Topic` in this resource, you need to define the `Topic` and add it to your `TopicList` like so (this is in the same class as above):
import org.globus.wsrf.impl.SimpleTopic;

...  
private static final QName topicName = new QName("http://www.globus.org/MyService", "MyTopic");

private void init(){
    Topic myTopic = new SimpleTopic(topicName);
    this.getTopicList().addTopic(myTopic);
}

There is nothing magical about the names or what method this occurs in. For simplicity's sake let's just assume that our Resource has an init method that will set up the Topic. This creates a new Topic with the supplied identifier and adds it to our list of Topics. Ok, our setup is done.

Java Core has provided a helper class to assist in subscribing to a Topic from a service. The SubscribeHelper provides a simple way for you service to subscribe a client to a Topic. You can use it like this:

```java
private EndpointReferenceType subscribe(QName topicName, ResourceKey key, Resource resource,
    EndpointReferenceType consumerAddr) throws ServiceException, RemoteException {
    Subscribe request = new Subscribe();
    request.setConsumerReference(consumerAddr);
    try {
        TopicExpressionType topicExpression = new TopicExpressionType();
        topicExpression.setDialect(WSNConstants.SIMPLE_TOPIC_DIALECT);
        topicExpression.setValue(topicName);
        MessageElement element =
            (MessageElement) ObjectSerializer.toSOAPElement(topicExpression, WSNConstants.TOPIC_EXPRESSION);
        FilterType filter = new FilterType();
        filter.set_any(new MessageElement[]{element});
        request.setFilter(filter);
        ResourceContext ctx = ResourceContext.getResourceContext();
        String homeLocation = ((ResourceContextImpl) ctx).getResourceHomeLocation();
        return new SubscribeHelper(
            ctx, resource, key, homeLocation, request).subscribe();
    } catch (Exception e) {
        throw new RemoteException("Error creating Subscription", e);
    }
}
```

This is a little ugly, but it is boilerplate code. Specify your topicName, resourceKey, resource, and the EndpointReferenceType of the client you are subscribing, and this will handle the subscription for you. You return the EndpointReferenceType of the subscription, so that the client can unsubscribe when they are complete.

Finally, once you have all of this set up the last piece is to actually send a message on your topic. This is pretty easy:

```java
LinkedList topicPath = new LinkedList();
topicPath.add(myTopicQName);
Topic myTopic = this.getTopicList().getTopic(topicPath);
```
myTopic.notify(myMessage);

This will send the notification to the Topic and, as a result, to all consumers for that topic.

It is not strictly necessary to use all of the ResourceProperties and TopicListAccessor Interfaces as shown above, but this is the scenario for which Java WS Core provides explicit support.

### 7. JNDI, Resources and Objects

These recipes all relate to using JNDI to store resources and objects to be used by your service.

#### 7.1. How do I access other objects from my service?

Services should avoid creating and destroying resources that live beyond the lifetime of a resource. However, this does not mean that shouldn't use objects that have a longer lifetime. Java WS Core uses JNDI to support these objects. Several objects already exist in JNDI, having been put there by Java WS Core.

- **commonj.work.WorkManager -- Constants.DEFAULT_WORK_MANAGER**
  
  This is a WorkManager for handling asynchronous work

- **commonj.timers.TimerManager -- Constants.DEFAULT_TIMER**
  
  This a timer for scheduling future or recurring asynchronous work

- **org.globus.wsrf.query.QueryEngine -- Constants.DEFAULT_QUERY_ENGINE**
  
  This is used to map queries on the service data of a service to the appropriate ExpressionEvaluators and then return the result.

- **org.globus.wsrf.topicexpression.TopicExpressionEngine -- Constants.DEFAULT_TOPIC_EXPRESSION_ENGINE**
  
  The TopicExpressionEngine interface is used to map queries on a topic list to the appropriate TopicExpressionEvaluator and then return the result.

- **javax.sql.DataSource -- Constants.DEFAULT_DATABASE**
  
  This is an in-memory base (Apache Derby), which can be used by services

To access these objects from JNDI, you can use a convenience method provided by Java WS Core.

```java
import org.globus.wsrf.jndi.JNDIUtils;
import org.globus.wsrf.Constants;

... 

WorkManager wm = (WorkManager) JNDIUtils.getInitialContext().lookup(Constants.DEFAULT_WORK_MANAGER);
```

Note, that you need to cast whatever you lookup to the proper type, which is, in this case, WorkManager. The org.globus.wsrf.Constants class has constant fields for looking up each of standard resources provided by Java WS Core.
### 7.2. How can I store my own resources in JNDI?

As stated before, services should avoid creating objects and resources that have a lifetime that extends beyond a single request directly, so service developers should create and store those objects in JNDI. Any object can be stored in JNDI. The simplest way to implement an object as a JavaBean, i.e., having properties with corresponding get/set methods. Let's assume we have a bean that has a timeout property. We can configure this to be put in JNDI by putting the following information in our jndi-config-deploy.xml file.

```xml
<resource name="MyBean" type="org.mypackage.MyBean">
  <resourceParams>
    <parameter>
      <name>timeout</name>
      <value>10000</value>
    </parameter>
  </resourceParams>
</resource>
```

This would set the timeout property to be 10000. All properties which follow the JavaBean pattern can be set this way.

If you want to work with a class that does not follow the JavaBean pattern, or that requires a greater level of set up, you can implement your own factory to be used by JNDI. The interface for ObjectFactory only has one method to be implemented

```java
Object getObjectInstance(Object obj, Name name, Context nameCtx, Hashtable environment)
```

The parameters to this method provide all of the information necessary to create your object. In most situations, you will only work with the first parameter.

```java
if (obj instanceof ResourceRef) {
    Reference ref = (Reference) obj;
    int timeout = ref.get("timeout");
    MyBean bean = new MyBean();
    bean.setTimeout(timeout);
    return bean;
} else{
    return null;
}
```

This is the most common way of creating an object factory. Assuming the first parameter is an instance of ResourceRef, we have a simple way of getting the parameters defined in the configuration. The get() method of ResourceRef allows us to access those parameters. Once you have your parameters, create and configure your object, then return it. Now you can use this object just like you could any of the resources that Java WS Core supplies for you. The lookup name is specified by the name attribute of the resource element in your configuration. To lookup an object you have supplied, you still use the JNDIUtils as before, specifying the proper lookup path.
MyBean bean = (MyBean) JNDIUtils.getInitialContext().lookup(Constants.DEFAULT_WSRF_LOCATION + "MyService/MyBean");

Any object you define in your jndi-config-deploy.xml file can be found at the default wsrf location followed by your service name and the name you specified in the jndi-config-deploy.xml file.

8. Asynchronous Work and Threads

8.1. How can I run work asynchronously from my service?

You can use the container to maintain your threads for you. This is considered the best approach towards dealing with concurrency. The container's thread management is handled by the WorkManager class. The WorkManager is an abstraction on top of how a task (or Work) is executed. It also assures that the execution context will be set correctly for the task. The abstract nature of the WorkManager provides a lot of flexibility to do things such as execute work in a pool, or on a remote cluster. To use the WorkManager, you need to construct a Work object. Work just extends the Runnable interface to supply a method which allows the Work to gracefully end its processing. Basically, though, you only need to implement a run() method, just like you normally would. One advantage the WorkManager provides is that the caller can get information about the progress of Work which has been submitted to the WorkManager. This includes the state, as well as any information regarding exceptions that might have been thrown. The WorkManager works in a similar way to the ExecutorServices.

import commonj.work.WorkManager;

...  
WorkManager manager = ... //acquired from JNDI  
manager.scheduleWork(myWork);

The above code will execute the myWork asynchronously in a separate thread. It is also possible to run work synchronously in a separate thread:

import commonj.work.WorkManager;

...  
WorkManager manager = ... //acquired from JNDI  
manager.doWork(myWork);

This will also execute the myWork in a separate thread, but it will block until the job has completed. The doWork method should be used sparingly.

8.2. How do I configure WorkManagers for my service?

Java WS Core sets up a default WorkManager which can be used by any service. The default WorkManager can be found in JNDI by doing the following:

11 http://edocs.beasys.com/wls/docs90/javadocs/commonj/work/WorkManager.html
WorkManager manager = (WorkManager) JNDIUtils.getInitialContext().lookup(Constants.DEFAULT_WORK_MANAGER);

(See the section on JNDI Objects for details about looking up the WorkManager)

However, the job queue and thread pool for the default WorkManager is shared amongst all of the services, and, as such, may not be appropriate for time-sensitive jobs. In addition, the default WorkManager has a simple default configuration, which may not meet the needs of a service. Most services will want to configure one or more WorkManagers for use by their service. To configure a WorkManager for your service, you will need to add a resource to your jndi-deploy-config.xml file (see the section on configuring JNDI for an overview)

There is only one parameter currently used for the WorkManager (more will be coming soon). This is the maximumPoolSize. This allows you to set the maximum number of threads that will be running at any given time. To configure a WorkManager for your service, you can specify the following:

```xml
<resource name="wm/WorkManager1" type="org.globus.wsrf.impl.work.WorkManagerImpl">
  <resourceParams>
    <parameter>
      <name>maximumPoolSize</name>
      <value>5</value>
    </parameter>
  </resourceParams>
</resource>
```

This will create a WorkManager that has a maximum size of 5 threads. Note that this is only the maximum number of threads. The WorkManager can still accept an unlimited number of jobs, but they will be queued until a thread is available to service them. This WorkManager can be acquired using the following code:

```java
WorkManager WorkManager1 = (WorkManager) JNDIUtils.getInitialContext().lookup(Constants.DEFAULT_WSRF_LOCATION + "/MyService/wm/WorkManager1");
```

Now the WorkManager can service your Work objects. Your service can use multiple WorkManagers. In fact, it is often advisable to have more than one WorkManager with a fairly small pool of threads (5-10) where each WorkManager processes a particular stage of your job. That way, long running tasks won't block fast tasks from executing.

### 8.3. How can I run a "Daemon" process from my service?

Sometimes a service will need to run a process that lasts for a very long time (for example, for the lifetime of the container). A typical example, would be a monitoring process that is constantly listening for some event to occur. Clearly, the service won't want one of the Threads from its WorkManager pool to be occupied with this permanent task. To resolve this, you can specify your Work object as a Daemon which will launch it in a new Thread automatically. You should still use the WorkManager for this task, since it sets the context for you and can reuse threads when a Daemon has completed. The only difference is that when you create your Work object, you will return true for the isDaemon() method.
8.4. How can I create a Thread from my service?

Creating and using Java threads is a complicated task because of classloader, security and transaction contexts. As such, it is strongly recommended that service developers not launch their own threads. If you, as a developer, feel you must launch your own thread, do so at your own risk. Here are a couple of pieces of advice, however if you are creating your own threads:

- **Use the java.utils.concurrent constructs**

  Java 5 has introduced a whole suite of concurrency tools which hide the details of handling threads. You should take advantage of these, specifically, the ExecutorService framework. This provides a way of abstracting thread pools and other thread handling.

  ```java
  import java.util.concurrent.Executors;
  import java.util.concurrent.ThreadPoolExecutor;
  ...
  //Create the Executor
  ExecutorService executor = Executors.newFixedThreadPool(numOfThreads);
  ...
  //Now use the executor
  executor.submit(myRunnable);
  ```

  This assumes that myRunnable is an instance of a class that implements Runnable. The above code will submit the Runnable to the executor, and when a thread becomes available, execute the Runnable. There are other options for creating your Executor service and if you have other needs, you should read the javadocs\(^\text{12}\). If you are going to use an ExecutorService, you should either set your service scope to "application", store the ExecutorService in JNDI or both. Otherwise, the ExecutorService will be created everytime your service has a request.

- **Use the ClassLoaderUtils from CoG**

  It is difficult to keep track of classloaders when you are working in threads. If you need to load a class using a method like `Class.forName(name)` or load a resource like `getResource(resource)` you must take special care to ensure that you are using the correct classloader. CoG provides a utility to help ensure that you are using the correct classloader. Instead of calling the above methods, use `ClassLoaderUtils.forName(name)` or `ClassLoaderUtils.getResource(resource)`. The ClassLoaderUtils has other helper methods as well. The documentation can be found [here]\(^\text{13}\).

Even with these tools, it is possible you may still run into problems with Threads, but these should help get you on the way.

9. Querying Properties

9.1. How can I query resource properties using XPath

Java WS Core currently provides two ways of querying the resource properties using XPath: the standard method and a proprietary method. The standard method is defined by the WS-ResourceProperties specification and all implementations of this specification support it. The proprietary method is a custom solution and therefore only supported

\(^\text{12}\) [http://java.sun.com/j2se/1.5.0/docs/api/java/util/concurrent/package-summary.html](http://java.sun.com/j2se/1.5.0/docs/api/java/util/concurrent/package-summary.html)

by Globus Toolkit. The proprietary method is a new query dialect called **TargetedXPath**. The **TargetedXPath** query dialect offers three key advantages over the standard XPath query method:

- **Namespace mappings** - a set of explicit namespace mappings to be passed along with the query. With these mappings the query expression can be dramatically simplified as namespace prefixes can be used freely within the query.

- **Single resource property querying** - a specific resource property can be queried instead of the entire resource property document.

- **WS-Enumeration support** - the query results can be returned as an enumeration.

The `globus-xpath-query` command line tool can be used to query resource properties with the **TargetedXPath** query dialect. If the query results were returned as an enumeration they can be retrieved using the `ws-enumerate` command line tool.

9.1.1. **API**

Example querying resource properties using the **TargetedXPath** query dialect:

```java
import org.globus.wsrf.core.query.targetedXPath.TargetedXPathQueryType;
import org.globus.wsrf.core.query.targetedXPath.NamespaceMappingType;
import org.globus.wsrf.query.targetedXPath.TargetedXPathConstants;

TargetedXPathQueryType targetedQuery = new TargetedXPathQueryType();
NamespaceMappingType nsMap[] = new NamespaceMappingType[1];
nsMap[0] = new NamespaceMappingType();
nsMap[0].setMappedName("fooPrefix"); nsMap[0].setNamespace(new URI("http://fooNamespace"));
targetedQuery.setNamespaceMappings(nsMap);
targetedQuery.setQueryString("boolean(//fooPrefix:fooElement)");
QueryExpressionType query = new QueryExpressionType();
query.setDialect(TargetedXPathConstants.TARGETED_XPATH_DIALECT);
query.setValue(targetedQuery);
QueryResourceProperties_Element queryRequest = new QueryResourceProperties_Element();
queryRequest.setQueryExpression(query);
QueryResourcePropertiesResponse queryResponse = port.queryResourceProperties(queryRequest);
```

To query a specific resource property do:

```java
...
TargetedXPathQueryType targetedQuery = new TargetedXPathQueryType();
targetedQuery.setNamespaceMappings(nsMap);
targetedQuery.setQueryString("boolean(//fooPrefix:fooElement)");
QName rp = new QName("http://foo", "bar");
targetedQuery.setResourcePropertyName(rp);
...
```

To return query results as an enumeration do:
import org.xmlsoap.schemas.ws._2004._09.enumeration.EnumerationContextType;
...

targetedQuery.setNamespaceMappings(nsMap);
targetedQuery.setQueryString("boolean(//fooPrefix:fooElement)");
targetedQuery.setEnumerateResults(Boolean.TRUE);
...

QueryResourcePropertiesResponse queryResponse = port.queryResourceProperties(queryRequest);
EnumerationContextType context =
    (EnumerationContextType)ObjectDeserializer.getFirstAsObject(queryResponse, EnumerationContextType.class);

9.2. How can I add a new query/topic expression evaluator?

Java WS Core allows for custom query/topic expression evaluators to be plugged in. The process of adding a new query/topic expression evaluator is composed of three steps:

9.2.1. Step 1: Implement the evaluator

Table 2.2. Evaluator interfaces

<table>
<thead>
<tr>
<th>If the evaluator is a...</th>
<th>then it must implement:</th>
</tr>
</thead>
<tbody>
<tr>
<td>query expression evaluator</td>
<td>org.globus.wsrfs.query.ExpressionEvaluator</td>
</tr>
<tr>
<td>topic expression evaluator</td>
<td>org.globus.wsrfs.topicexpression.TopicExpressionEvaluator</td>
</tr>
</tbody>
</table>

9.2.2. Step 2: Register the evaluator

The evaluators must be registered in order for Java WS Core to recognize them. The registration is done through the JNDI configuration file. The expression evaluators must be deployed as global resources under a specific subcontext.

9.2.2.1. Registering query expression evaluators

The query expression evaluators must be deployed as global resources under the query/eval/ subcontext in the JNDI configuration file.

Example:

```
<global> <resource name="query/eval/MyQueryExpressionEval" type="foo.bar.MyQueryExpressionEvaluator">
    <resourceParams> <parameter> <name>factory</name> <value>org.globus.wsrfs.jndi.BeanFactory</value> </parameter> </resourceParams> </resource>
</global>
```

Where the <resource> attribute:
<table>
<thead>
<tr>
<th>name</th>
<th>Specifies the name of the evaluator in JNDI space. The name can be arbitrary as long as it is unique and is in the right subcontext as explained above.</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Specifies the class that implements the expression evaluator.</td>
</tr>
</tbody>
</table>

### 9.2.2.2. Registering topic expression evaluators

Topic expression evaluators must be deployed as global resources under the `topic/eval/` subcontext in the JNDI configuration file.

Example:

```xml
<global>  
  <resource name="topic/eval/MyTopicExpressionEval" type="foo.bar.MyTopicExpressionEvaluator">  
    <resourceParams>  
      <parameter>  
        <name>factory</name>  
        <value>org.globus.wsrf.jndi.BeanFactory</value>  
      </parameter>  
    </resourceParams>  
  </resource>  
</global>
```

Where the `<resource>` attribute:

<table>
<thead>
<tr>
<th>name</th>
<th>Specifies the name of the evaluator in JNDI space. The name can be arbitrary as long as it is unique and is in the right subcontext as explained above.</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Specifies the class that implements the expression evaluator.</td>
</tr>
</tbody>
</table>

### 9.2.3. Step 3: Register the serializer/deserializer for the evaluator

A serializer/deserializer must be registered for the dialect of the evaluator in order for the expression to be properly serialized and deserialized. The serializers/deserializers for the dialect are deployed as almost any other type mapping. In general, each type mapping specifies a type QName. For dialect serializers/deserializers, that type QName takes a slightly different name.

#### 9.2.3.1. Specifying the QName for query expression evaluators

For query expression evaluators, that QName must have the local name part set to `QueryExpressionDialect` and namespace part set to the dialect of the query expression evaluator.

Example:

```xml
<typeMapping encodingStyle=""  
  serializer="org.apache.axis.encoding.ser.SimpleSerializerFactory"  
  deserializer="org.apache.axis.encoding.ser.SimpleDeserializerFactory"  
  type="java:java.lang.String"  
  qname="ns12:QueryExpressionDialect"  
  xmlns:ns12="http://foo.bar/MyQueryDialect"/>
```

⚠️ **Note**

These type mappings must be deployed both on the client and the server.
9.2.3.2. Specifying the QName for topic expression evaluators

For topic expression evaluators, that QName must have the local name part set to `TopicExpressionDialect` and namespace part set to the dialect of the topic expression evaluator.

Example:

```xml
```

Note

These type mappings must be deployed both on the client and the server.

9.2.4. Step 4: Configuring a helper serializer for GetCurrentMessage-Provider

The standard `GetCurrentMessageProvider` might not know how to properly serialize the notification message currently associated with the specified topic. The `GetCurrentMessageProvider` can be configured to use a helper serializer for a given notification message type.

To configure such a helper serializer, define the following global resource in your `deploy-jndi.xml` configuration file:

```xml
```

Where the `<resource>` attribute:

<table>
<thead>
<tr>
<th>name</th>
<th>Must start with <code>providers/GetCurrentMessageProvider/</code> and must end with the full class name of the notification message.</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Specifies the class that implements the <code>org.globus.wsrf.encoding.ObjectConverter</code> interface and is responsible for serializing the notification message. The <code>GetCurrentMessageProvider</code> will use the type of the notification message to find the helper serializer.</td>
</tr>
</tbody>
</table>

10. Build Recipes

10.1. How can I process my wsdl using an Ant build file?

Java WS Core supplies some helper build files to simplify the task of processing wsdl. In order to use the helper build files, you should tell ant about GLOBUS_LOCATION. You can set an Ant property to do this:
<property environment="env"/>
<property name="globus_location" value="\${env.GLOBUS_LOCATION}"/>

The first property statement creates a property which holds all of the system environment variables and the second assigns the GLOBUS_LOCATION environment variable to the property globus_location. Obviously, this means you need to have set the GLOBUS_LOCATION variable in your environment. Once you have set this, you can use this property in your target.

If you have used the wsdlpp:extends (where wsdlpp is http://www.globus.org/namespaces/2004/10/WSDLPreprocessor) property in your wsdl, the first step to preparing your stubs is to flatten your wsdl file. To run this task, use this:

```xml
<ant antfile="\${build.stubs}" target="flatten">
   <property name="source.flatten.dir" location="\${schema.src}/myservice"/>
   <property name="target.flatten.dir" location="\${schema.dest}/myservice"/>
   <property name="wsdl.source" value="myservice_port.wsdl"/>
   <property name="wsdl.target" value="myservice_flattened.wsdl"/>
   <property name="wsdl.porttype" value="MyServicePortType"/>
</ant>
```

The arguments here are fairly self-explanatory, but just to clarify the source.flatten.dir represents the directory that your wsdl file is in, the target.flatten.dir is the directory that you want your processed wsdl file to be written to, wsdl.source is the name of the wsdl file that you are processing, wsdl.target is the name that you want the new wsdl file to be called, and wsdl.porttype is the port type that you are processing (as defined in your wsdl file. You'll notice that there are references to properties such as \${build.stubs}, \${schema.src}, etc. These are just properties I set up. You can set these up to whatever you wish (or just use the paths directly instead of using properties).

The next step is to generate the bindings for your service. This is very similar to the previous step:

```xml
<target name="binding" depends="flatten">
   <ant antfile="\${build.stubs}" target="generateBinding">
      <property name="source.binding.dir" location="\${schema.dest}/myservice"/>
      <property name="target.binding.dir" location="\${schema.dest}/myservice"/>
      <property name="binding.root" value="myservice"/>
      <property name="porttype.wsdl" value="myservice_flattened.wsdl"/>
   </ant>
```

The arguments here are fairly self-explanatory, but just to clarify the source.flatten.dir represents the directory that your wsdl file is in, the target.flatten.dir is the directory that you want your processed wsdl file to be written to, wsdl.source is the name of the wsdl file that you are processing, wsdl.target is the name that you want the new wsdl file to be called, and wsdl.porttype is the port type that you are processing (as defined in your wsdl file. You'll notice that there are references to properties such as \${build.stubs}, \${schema.src}, etc. These are just properties I set up. You can set these up to whatever you wish (or just use the paths directly instead of using properties).
Again, the arguments should be fairly clear. The source.binding.dir is the directory where the wsdl that you want to bind resides (this will almost always be the target.flatten.dir from the previous step). The target.binding.dir is the directory where the bindings will be written. The binding.root is the base name of the outputted wsdl files, and the porttype.wsdl is the name of the wsdl file to bind (This will almost always be the name of the wsdl.target from the previous step).

The final step to prepare your wsdl is to generate the stub classes. These are the java classes to which you will refer in your service and your clients. Similar to the previous two steps:

```xml
<target name="stubs" depends="binding">
  <ant antfile="${build.stubs}" target="mergePackageMapping">
    <property name="mapping.src" value="${basedir}/namespace2package.properties"/>
    <property name="mapping.dst" value="${build.dir}/namespace2package.properties"/>
  </ant>

  <ant antfile="${build.stubs}" target="generateStubs">
    <property name="source.stubs.dir" location="${schema.dest}/test/performance"/>
    <property name="target.stubs.dir" location="${stubs.src}"/>
    <property name="wsdl.file" value="performance_test_service.wsdl"/>
    <property name="mapping.file" value="${build.dir}/namespace2package.properties"/>
  </ant>
</target>
```

There are actually two Globus-provided tasks in this target. The first one is optional. If you have custom mappings to map namespaces to Java package names, you can create a properties file that lists those mappings. Then use the mergePackageMapping task to merge your mappings with the default mappings. The parameters are: mapping.src -- where your custom mappings live and mapping.dst -- where you want the merged file to be written.

The final task here (generateStubs) actually generates the stub classes. The parameter names here are not entirely intuitive. source.stubs.dir is the directory where your wsdl's live. target.stubs.dir is where you want the Java source files to be written. wsdl.file is the name of your wsdl file, and mapping.file is the location of the (optional) namespace-package mapping file.

Once you have run all of these steps, you should generated all of the java stubs and bindings that you need to build your service.

### 10.2. How can I create a GAR?

To create a GAR file use the following example:

```xml
<property name="build.packages" location="${deploy.dir}/share/globus_wsrf_common/build-packages.xml"/>
...
```
<property name="garjars.id" value="garjars"/>
<fileset dir="lib" id="garjars"/>
<property name="garetc.id" value="garetc"/>
<fileset dir="etc" id="garetc"/>
...

<target name="dist" depends="...">
  <ant antfile="${build.packages}" target="makeGar">
    <property name="gar.name" value="mygar.gar"/>
    <reference refid="${garjars.id}"/>
    <reference refid="${garetc.id}"/>
  </ant>
</target>

The `gar.name` property must be passed. That property specifies the gar file to create. The `makeGar` task will look for `deploy-client.wsdd`, `deploy-server.wsdd`, and `deploy-jndi-config.xml` files in the base directory of the calling Ant process. All of these files are optional and do not have to exist. The list of files to be included in the GAR file is passed via Ant references. The `makeGar` accepts the following references: `garjars.id`, `garschema.id`, `garetc.id`, `garshare.id`, `gardocs.id`, and `garbin.id`. All of these references are optional and do not have to be defined.

In the above example, all files in the `etc` and `lib` directories, and the `deploy-client.wsdd`, `deploy-server.wsdd`, and `deploy-jndi-config.xml` files (if they exist) will be included into the GAR file.

10.3. How can I deploy my GAR?

To deploy a GAR file use the following example:

```xml
<property name="build.packages" location="${deploy.dir}/share/globus_wsrf_common/build-packages.xml"/>
...
<target name="deploy" depends="...">
  <ant antfile="${build.packages}" target="deployGar">
    <property name="gar.name" value="mygar.gar"/>
  </ant>
</target>
```

The `gar.name` property must be passed. That property specifies the gar file to deploy. Optionally, the `profile` property can be passed to indicate which configuration profile the gar should be deployed under.

10.4. How can I generate a laucher for my client?

Bourne Shell and Windows batch scripts can be automatically generated to hide the details of launching a Java program from the command line.

To generate such a command line script, write a Ant task that calls the `generateLauncher` target in `$GLOBUS_LOCATION/share/globus_wsrf_common/build-launcher.xml`. The following properties/parameters must be specified:
• $\{launcher-name\}$ - the base name of script to generate.

• $\{class.name\}$ - the name of Java class the script must call.

For example:

```xml
<property name="env.GLOBUS_LOCATION" value="."/>
<property name="deploy.dir" location="${env.GLOBUS_LOCATION}"/>
<property name="abs.deploy.dir" location="${deploy.dir}"/>
<property name="build.launcher" location="${abs.deploy.dir}/share/globus_wsrf_common/build-launcher.xml">

<ant antfile="${build.launcher}" target="generateLauncher">
  <property name="launcher-name" value="myClient"/>
  <property name="class.name" value="org.mypackage.MyClient"/>
</ant>
```

It is also possible to specify default JVM options and command line options via the `default.jvm.options` and `default.cmd.line` parameters. When passing multiple parameters using `default.jvm.options` for Unix/Linux scripts the parameters must be separated by $(DELIM) delimiter. For example:

```xml
<target name="generateUnixScripts" if="generate.unix" depends="testUnix">
  <ant antfile="${build.launcher}" target="generateLauncher">
    <property name="default.jvm.options" value="-DFOO="$FOO"${DELIM}-DBAR="$BAR"/>
  </ant>
</target>
```

In general the generation of the command line scripts is done in the `post-deploy.xml` script during GAR deployment (globus-deploy-gar).

10.5. How do I create a client distribution for my service?

There is no automated process for this, but there are a few simple steps that can provide the pieces that you need. (Note, this is one possible approach. Other methods can work too.) First, when you are building your service, you need to generate your stubs in their own directory. From Ant, this would look like this (from the CounterService example):

```xml
<target name="stubs" unless="stubs.present" depends="init">
  <ant antfile="${build.stubs}" target="generateStubs">
    <property name="source.stubs.dir" 31
```
The `source.stubs.dir` determines where the stub classes are created. This, in turn, will allow us to compile those classes into their own compile directory like this (also from the CounterService example):

```xml
<target name="compileStubs" depends="stubs">
    <javac srcdir="${stubs.src}" destdir="${stubs.dest}"
        debug="${java.debug}">
        <include name="**/*.java"/>
        <classpath>
            <fileset dir="${deploy.dir}/lib">
                <include name="common/*.jar"/>
                <include name="*.jar"/>
                <exclude name="${stubs.jar.name}"/>
                <exclude name="${jar.name}"/>
            </fileset>
        </classpath>
    </javac>
    <copy todir="${build.dest}" >
        <fileset dir="src" includes="**/*.properties" />
        <fileset dir="src" includes="**/*.xml" />
    </copy>
</target>
```

So, now we have our stubs compiled into their own directory and copied all resource files into the same directory. The final step in preparing the stubs is to jar them up:

```xml
<target name="jarStubs" depends="compileStubs">
    <jar destfile="${build.lib.dir}/${stubs.jar.name}"
        basedir="${stubs.dest}"/>
</target>
```

At this point, we have a jar which contains all of our stubs. This will be used on both the server and by the clients. Now, you need to write your client code (probably in its own src directory), compile it and create a jar for it. These two jars are the only jars you need to produce. In addition you need to make sure that you have an installation of Java WS Core available. The root directory of your installation should have both a `client-config.wsdd` and a `log4j.properties` file in it. You can combine this with the launcher example above to build a launcher for the client jars which you have just generated.
11. Controlling client connection settings

By default Java WS Core clients will use HTTP 1.1 protocol with chunked encoding. Java WS Core clients will also attempt to reuse HTTP/S connections between the calls. The default timeout for clients is 10 minutes. All of these connection properties can be controlled programmatically using the HTTPUtils API as shown below.

**Note**

Please note that once a connection property is set on a given Stub, it is applied to ALL calls made using that Stub.

### 11.1. How can I set the timeout on my client?

To set connection timeout do (the timeout value is in milliseconds):

```java
import org.globus.axis.transport.HTTPUtils;
...

MyServiceAddressingLocator locator = new MyServiceAddressingLocator();
MyService port = locator.getMyServicePort(url); // set timeout to 2 minutes
HTTPUtils.setTimeout((Stub)port, 1000 * 60 * 2);
port.hello();
```

### 11.2. How can I reuse my connection?

To control connection reuse do:

```java
import org.globus.axis.transport.HTTPUtils;
...

MyServiceAddressingLocator locator = new MyServiceAddressingLocator();
MyService port = locator.getMyServicePort(url); // close connection after the call
HTTPUtils.setCloseConnection((Stub)port, true);
port.hello();
// do not close connection - let it be reused
HTTPUtils.setCloseConnection((Stub)port, false);
port.hello();
```

### 11.3. How can I control HTTP chunked encoding

To control whether HTTP chunked encoding should be used do:

```java
import org.globus.axis.transport.HTTPUtils;
```

---

MyServiceAddressingLocator locator = new MyServiceAddressingLocator();
MyService port = locator.getMyServicePort(url); // disable chunked encoding
HTTPUtils.setChunkedEncoding((Stub)port, false);
port.hello();
// re-enable chunked encoding
HTTPUtils.setChunkedEncoding((Stub)port, true);
port.hello();
11.4. How can I control the HTTP protocol version?
To control what HTTP protocol version should be used do:

import org.globus.axis.transport.HTTPUtils;

MyServiceAddressingLocator locator = new MyServiceAddressingLocator();
MyService port = locator.getMyServicePort(url);
// force HTTP 1.0
HTTPUtils.setHTTP10Version((Stub)port, true); port.hello();
// force HTTP 1.1
HTTPUtils.setHTTP10Version((Stub)port, false);
port.hello();

12. Writing and running tests
One of the most important tasks in developing services is to create tests to assure that your code functions correctly. The following recipes discuss creating tests for Java WS Core.

12.1. What is JUnit and how does it relate to Java WS Core?
Tests in the Java WS Core are based on the JUnit\(^\text{15}\) API. JUnit must first be installed with Ant. To install JUnit with Ant copy the junit.jar found in JUnit distribution to the $ANT_HOME/lib directory. Alternatively, you can add the junit.jar to your CLASSPATH, or source $GLOBUS_LOCATION/etc/globus-devel-env.sh.

12.2. How do I write JUnit tests for services?
Always make sure to group your tests under the PackageTests.java and/or SecurityTests.java test suites. Put all tests that require any type of credentials in the SecurityTests.java test suite.

If you are writing basic unit tests that do not require a container to run, just use the regular JUnit classes to write such tests.

\(^{15}\) http://www.junit.org/
If you are writing tests that require a container to execute, use the `org.globus.wsrf.test.GridTestCase` class instead of `junit.framework.TestCase` as your base class for your tests. Also ensure your `PackageTests.java` or `SecurityTests.java` extends the `org.globus.wsrf.test.GridTestSuite` instead of `junit.framework.TestSuite`.

The `org.globus.wsrf.test.GridTestSuite` and `org.globus.wsrf.test.GridTestCase` must be used together. The `org.globus.wsrf.test.GridTestCase` class exposes a `TEST_CONTAINER` variable that can be used to obtain the URL of the container (`TEST_CONTAINER.getBaseURL()`). By default an embedded container will be started for all tests in the test suite. To specify an external container, pass the `-Dweb.server.url=<base.url>` system property on the `java` command line.

### 12.3. How do I run my JUnit tests for Java WS Core and/or my services?

#### 12.3.1. Group testing

To execute all tests contained in a given jar file with an internal container run the following:

```bash
$ cd $GLOBUS_LOCATION $ ant -f share/globus_wsrf_test/runtests.xml run
-Dtests.jar=<test.jar>
```

Where `<test.jar>` is an absolute path to the jar file that contains the tests.

To execute all tests contained in a given jar file with an external container run the following:

```bash
$ cd $GLOBUS_LOCATION $ ant -f share/globus_wsrf_test/runtests.xml runServer
-Dtests.jar=<test.jar>
```

By default, the external container is assumed to be running at `http://localhost:8080/wsrf/services/`. To specify a different container, use the `-Dtest.server.url=<url>` property.

By default, all `PackageTests` and `SecurityTests` tests will be executed. To execute `PackageTests` only, specify `-DbasicTestsOnly=true` option. To execute `SecurityTests` only, specify `-DsecurityTestsOnly=true` option.

By default, the test results will be generated in the XML format.

#### 12.3.2. How do I run a single test?

To execute a single test suite with an internal container run the following:

```bash
$ cd $GLOBUS_LOCATION $ ant -f share/globus_wsrf_test/runtests.xml runOne
-Dtest.class=<test.class>
```

Where `<test.class>` is a Java class that contains a test suite.

To execute a single test suite with an external container run the following:

```bash
$ cd $GLOBUS_LOCATION $ ant -f share/globus_wsrf_test/runtests.xml runOneServer
-Dtest.class=<test.class>
```

By default, the external container is assumed to be running at `http://localhost:8080/wsrf/services/`. To specify a different container, use the `-Dtest.server.url=<url>` property.
By default, the test results will be generated in the plain text format.

It is also possible to execute a single test case (or a set of test cases) within a test suite by specifying a `-Dtests=<testCase1[,testCaseN]>` property. However, this will only work with test suites that inherit from `org.globus.wsrfs.test.FilteredTestSuite` or `org.globus.wsrfs.test.GridTestSuite` classes. Example:

```
$ cd $GLOBUS_LOCATION $ ant -f share/globus_wsrfs_test/runtests.xml runOne \ 
-Dtest.class=org.globus.interop.widget.test.PackageTests \ 
-Dtests="testScenario1,testScenario2"
```

### 12.3.3. How do I configure test output and options?

The test reports will be put in the `$GLOBUS_LOCATION/share/globus_wsrfs_test/tests/test-reports` directory by default. A different test reports directory can be specified by passing `-Djunit.reports.dir=<directory>`. Use `-Djunit.test.format` property to generate the test results in a specified format (xml or plain). Example:

```
$ ant -f share/... -Djunit.test.format=plain
```

Use `-Djunit.jvmarg` to pass arbitrary properties to the testing JVM. Example:

```
$ ant -f share/... 
-Djunit.jvmarg="-Dorg.globus.wsrfs.container.server.id=myServerID"
```

### 13. Miscellaneous Recipes

#### 13.1. How do I obtain Version Information about Java WS Core

The `Version` API can be used to obtain Java WS Core version information programmatically. For example to display major, minor and patch version information do:

```
import org.globus.wsrfs.utils.Version;
...

System.out.println("Major: " + Version.getMajor());
System.out.println("Minor: " + Version.getMinor());
System.out.println("Micro: " + Version.getMicro());
```

#### 13.2. How can I use SOAP Attachments with my service?

Java WS Core supports SOAP with Attachments. DIME, MIME, and MTOM formats are supported. This section provides brief sample code. Detailed code can be found in the automated tests for this feature at, [http://viewcvs.globus.org/](http://viewcvs.globus.org/)

---

16 [http://www.globus.org/api/javadoc-4.2.0/org/globus/wsrf/utils/Version.html](http://www.globus.org/api/javadoc-4.2.0/org/globus/wsrf/utils/Version.html)
To add an attachment to a request do:

```java
import javax.activation.DataHandler;
import javax.activation.FileDataSource;
import javax.xml.rpc.Stub;
import org.apache.axis.client.Call;
....
File file = new File(..);
DataHandler dataHandler = new DataHandler(new FileDataSource(file));
((Stub)port)._setProperty(Call.ATTACHMENT_ENCAPSULATION_FORMAT, Call.ATTACHMENT_ENCAPSULATION_FORMAT_MTOM);
((org.apache.axis.client.Stub)port).addAttachment(dataHandler);
```

To retrieve attachments associated with a request do:

```java
import javax.activation.DataHandler;
import javax.xml.soap.AttachmentPart;
import org.apache.axis.Message;
import org.apache.axis.MessageContext;
....
MessageContext msgContext = MessageContext.getCurrentContext();
Message reqMsg = msgContext.getRequestMessage();
if (reqMsg.getAttachmentsImpl() == null) {
    throw new Exception("Attachments are not supported");
}
Iterator it = reqMsg.getAttachments();
while (it.hasNext()) {
    AttachmentPart part = (AttachmentPart) it.next();
    DataHandler dataHandler = part.getDataHandler();
}
```

The SwA support adds a small overhead to overall SOAP processing. To disable SwA support for improved performance delete `$GLOBUS_LOCATION/lib/common/geronimo-activation_1.0.2_spec-1.1-SNAPSHOT.jar` files.

Please note that SOAP attachments can be used with message security but they will not be signed or encrypted.

### 13.3. How can I make sure my WSDL/SOAP documents are valid for GT 4?

The WSRF and WSN specifications schemas follow the document/literal mode as described in WS-I Basic Profile. The Basic Profile defines certain rules to follow for document/literal and other modes to ensure interoperability.

Java WS Core relies on these restrictions so please keep them in mind when designing your own schema.
13.3.1. Document/literal

In the document/literal mode as defined in the WS-I Basic Profile at most one <wsdl:part> is allowed in the <wsdl:message> element and it must use the 'element' attribute. Also, the wire signatures must be unique (cannot use the same 'element' attribute in <wsdl:part> in two different <wsdl:message> elements).

Note

Axis' WSDL2Java tool might sometimes incorrectly detect that schema follows the wrapped/literal mode and generate wrong stub and type classes. To ensure that document/literal mode is always used:

- use Java WS Core's generateStub* Ant tasks in <install>/share/globus_ws-rf_tools/build-stubs.xml file
- if you are using Axis' WSDL2Java tool directly, you can alternatively specify the -W command line option.

Also, with wrapped/literal mode, the element name had to match the operation name in wsdl. This is not necessary with document/literal mode.

13.3.2. SOAP Encoding

Do not use or mix the literal mode with the SOAP encoding mode (R2706)\(^\text{17}\). For example, do not use the soapenc:Array type. Please see the 5.2.3 section\(^\text{18}\) in the WS-I Basic Profile for details.

13.4. How can I retrieve service parameters from my WSDD file?

While we strongly recommend that you use the JNDI mechanism to provide your service with configuration information, it is sometimes necessary to obtain the value of parameters set in the WSDD file although it is extremely rare. If you aren't sure that you need to do this, you likely do not. Java WS Core provides some helper functions to ease this process:

```java
import org.globus.wsrf.utils.ContextUtils;
import org.apache.axis.MessageContext;

... 
MessageContext context = MessageContext.getCurrentContext();
String sampleProperty = (String) ContextUtils.getServiceProperty(context, "myProperty");
...
```

Note that this function requires that a MessageContext is associated with the current thread, which in general means that the call needs to happen within the context of a web service invocation.

\(^\text{17}\) http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html#refinement16638080
\(^\text{18}\) http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html#refinement16556272
Note

Specifying parameters using WSDD files depends on Axis and will likely not be supported in future versions of the toolkit.

13.5. How do I get standard MessageContext properties?

The following properties can be obtained from the SOAPContext/MessageContext associated with the current thread:

- `org.apache.axis.Constants.MC_HOME_DIR` - the base directory from which the wsdl files are loaded.
- `org.apache.axis.Constants.MC_CONFIGPATH` - the base directory from which different configuration files are loaded.

The `Constants.MC_CONFIGPATH` property should be used to load any type of configuration file. Only `Constants.MC_CONFIGPATH` and `Constants.MC_HOME_DIR` are associated with the thread during activation. In the standalone container the `Constants.MC_HOME_DIR` and `Constants.MC_CONFIGPATH` properties will usually point to the same directory. However, in Tomcat they will point to two different directories. Since GT 4.0.1, the `Constants.MC_HOME_DIR` value can be accessed using the `org.globus.wsrf.ContainerConfig.getSchemaDirectory()` static call, and `Constants.MC_CONFIGPATH` value via the `org.globus.wsrf.ContainerConfig.getBaseDirectory()` static call.
Chapter 3. Tutorials and Presentations

1. Tutorials
   
   • The Globus Toolkit 4 Programmer's Tutorial by Borja Sotomayor
   
   • Basic Grid Development Tools Tutorial
   
   • GT3.2 to GT4 Migration: A First HOWTO by Terry Harmer and Julie McCabe

2. Presentations
   
   • Java WS Core for Developers (Session at GlobusWorld 2006)

1 http://gdp.globus.org/gt4-tutorial/
2 http://ds.informatik.uni-marburg.de/MAGE/gdt/tutorial.html
3 http://www.qub.ac.uk/escience/howtos/GT3%20to%20GT4%20Version%200.3.htm
4 http://www.globus.org/toolkit/presentations/JavaWSCore-GW.ppt
Chapter 4. Samples

1. Counter Sample

Using counter-client.

1. Change to $GLOBUS_LOCATION directory:
   
   $ cd $GLOBUS_LOCATION

2. Start the container by running:
   
   $ bin/globus-start-container -nosec

3. In another window, run:
   
   $ bin/counter-client -s http://localhost:8080/wsrf/services/CounterService
   
   As a result you should see something like the following:
   
   $ bin/counter-client -s http://localhost:8080/wsrf/services/CounterService
   
   Counter service: http://localhost:8080/wsrf/services/CounterService Got
   
   with value: 3 Counter has value: 3 Got notification with value: 13

   Please note if secure container is used (started without the -nosec argument) please make sure to pass the correct service url (https protocol, right port number) to counter-client and add the -z argument at the end of the command line. For example:
   
   $ bin/counter-client -s https://localhost:8443/wsrf/services/CounterService
   
   The -z none option disables client-side authorization. By default the client performs host authorization and if the server is not running with host credentials the client authorization will fail. Also, the server must be properly configured to authorize the client. Please see the security documentation for details.

Using counter-create, counter-add clients.

1. Change to the $GLOBUS_LOCATION directory:
   
   $ cd $GLOBUS_LOCATION

2. Start the container by running:
   
   $ bin/globus-start-container -nosec

3. In another window, run:
   
   $ bin/counter-create -s
   
   http://140.221.36.11:8080/wsrf/services/CounterService > epr

   If successful, a new counter resource will be created and the endpoint information of that resource will be saved in a file called epr. The epr file can be passed to a number of the basic wsrf-* and wsn-* clients using the -e option.
Please note if the secure container is used (started without the \-nosec argument) to make sure to pass the correct service url (https protocol, right port number) to counter-create and add the \-z argument to the command line. For example:

```
$ bin/counter-create \-s https://localhost:8443/wsrf/services/CounterService \-z none > epr
```

4. In the same window, run (a couple of times):

```
$ bin/counter-add \-e epr 2
```

As a result you should see something like the following:

```
$ bin/counter-add \-e epr 2 2 $ bin/counter-add \-e epr 2 4
```

Please note that if secure container was used you might need to add the \-z argument to the command line. For example:

```
$ bin/counter-add \-e epr \-z none 2
```

## 2. Management Sample

The ManagementService sample service allows the users to view and dynamically modify the WSDD properties of a given service. The WSDD information of a given service is exposed as resource properties. In this example, we will be removing the QueryResourceProperties operation provider from the ContainerRegistryService.

Please note that the changes made to the services via the ManagementService are not permanent.

1. Change to the $GLOBUS_LOCATION directory:

```
$ cd $GLOBUS_LOCATION
```

2. Start the container by running:

```
$ bin/globus-start-container
```

3. In another window, run:

```
$ bin/wsrf-query \-z self \-s https://localhost:8443/wsrf/services/ManagementService \-k {http://axis.org}ServiceName ContainerRegistryService
```

As a result you should see something like the following:

```
<ns1:ServiceWSDD xmlns:ns0="http://xml.apache.org/axis/wsdd/"
    xmlns:ns1="http://xml.apache.org/axis/wsdd/">
  <ns1:loadOnStartup>true</ns1:loadOnStartup>
  <ns1:providers>GetRPProvider</ns1:providers>
  <ns1:providers>GetMRPProvider</ns1:providers>
  <ns1:providers>QueryRPProvider</ns1:providers>
  <ns1:handlerClass>org.globus.axis.providers.RPCProvider</ns1:handlerClass>
  <ns1:className>org.globus.registry.RegistryService</ns1:className>
  <ns1:allowedMethodsClass>org.globus.core.registry.RegistryPortType</ns1:allowedMethodsClass>
```
The above command queries the ManagementService and returns the WSDD information of the ContainerRegistryService.

4. To demonstrate that the ContainerRegistryService supports the QueryResourceProperties operation run the following command:

```bash
$ bin/wsrf-query -z self -s https://localhost:8443/wsrf/services/ContainerRegistryService
```

As a result you should see something like the following (a list of `<ns1:Entry>`):

```xml
<ns0:RegistryRP xmlns:ns0="http://www.globus.org/namespaces/2004/06/registry">
  <ns1:Entry> ... </ns1:Entry> <ns1:Entry>
  ... </ns1:Entry> </ns1:RegistryRP>
```

5. Next, create an `in.xml` file with the following contents:

```xml
<ns1:doc xmlns:ns1="http://xml.apache.org/axis/wsdd/">
  <ns1:providers>GetRPProvider</ns1:providers>
  <ns1:providers>GetMRFProvider</ns1:providers>
</ns1:doc>
```

Then run the following command:

```bash
$ bin/wsrf-update-property -z self -s https://localhost:8443/wsrf/services/ManagementService \
  -k {http://axis.org}ServiceName ContainerRegistryService in.xml
```

The above command updates the providers resource property of the ContainerRegistryService resource of ManagementService to remove the QueryRPProvider provider to be removed from the operation provider list of the ContainerRegistryService.

6. Now perform the query operation again:

```bash
$ bin/wsrf-query -z self -s https://localhost:8443/wsrf/services/ContainerRegistryService
```

Since the operation provider for the QueryResourceProperties operation was removed the above command should generate the following error:

```
Error: java.lang.Exception: [CORE] Operation 'QueryResourceProperties' defined in wsdl but it's not implemented in the 'ContainerRegistryService'
```

You can add the QueryResourceProperties operation provider by adding the `<ns1:providers>QueryRPProvider</ns1:providers>` entry to the `in.xml` file and running the command in step 5 again.
Chapter 5. Architecture and design overview

- Java WS Core Design Document [doc\(^1\) | pdf\(^2\)]
  - Java WS Core UML [vsd\(^3\) | Resource (.gif)\(^4\), Resource Property (gif)\(^5\), Notification (gif)\(^6\)]
  - Java WS Core Notification UML Sequence Diagrams [vsd\(^7\) | Subscription (jpg)\(^8\), Notification (jpg)\(^9\)]
- Java WS Core Container Dynamic Deploy Design Document [doc\(^10\) | pdf\(^11\)]
- Java WS Core WS-Enumeration Design Document [doc\(^12\) | pdf\(^13\)]

---

1. JavaWSCoreDesign.doc
2. JavaWSCoreDesign.pdf
3. JavaWSCoreUML.vsd
4. Resource.gif
5. ResourceProperty.gif
6. Notification.gif
7. NotificationSequences.vsd
8. NotificationSequence1.jpg
9. NotificationSequence2.jpg
10. HotDeployDesign.doc
11. HotDeployDesign.pdf
12. WSEnumDesign.doc
13. WSEnumDesign.pdf
Chapter 6. APIs

1. Programming Model Overview

There are two main parts to the Java WS Core programming model: the service and the resource. The service performs the business logic on the resource, and the resource represents the managed state.

The web service is just a stateless POJO (Plain Old Java Object) that discovers the resource associated with the request and then performs operations on that resource.

The resources are managed and discovered via ResourceHome implementations. The ResourceHome implementations can also be responsible for creating new resources, performing operations on a set of resources at a time, etc.

The ResourceHome implementations are configured in JNDI and are associated with a particular web service. JNDI is a central transient registry that is mainly used for discovery of ResourceHome implementations but it can be used to store and retrieve arbitrary information.

The web services are configured in the Web Services Deployment Descriptor (WSDD).

WSDL in document/literal style is assumed as a starting point for writing a service. A number of generic interfaces are defined so that custom implementations of these interfaces can be used instead of the default implementations included in the Java WS Core. No special base classes are required for a web service or resource implementation. However, the resource object at minimal must implement the org.globus.wsrf.Resource interface (it defines no operations; it is used just as a marker interface). The service developer can pick and choose which other resource interfaces the resource object should implement to tailor the implementation to his/her needs.

2. Component API

- Stable API:
  - org.globus.wsrf.Resource
  - org.globus.wsrf.ResourceKey
  - org.globus.wsrf.ResourceProperty
  - org.globus.wsrf.ResourcePropertyMetaData
  - org.globus.wsrf.ResourcePropertySet
  - org.globus.wsrf.ResourceProperties
  - org.globus.wsrf.ResourceHome
  - org.globus.wsrf.ResourceLifetime
  - org.globus.wsrf.ResourceIdentifier
  - org.globus.wsrf.ResourceContext
  - org.globus.wsrf.RemoveCallback
  - org.globus.wsrf.PersistentCallback
- org.globus.wsrf.Subscription
- org.globus.wsrf.Topic
- org.globus.wsrf.TopicList
- org.globus.wsrf.TopicListMetaData
- org.globus.wsrf.TopicAccessor
- org.globus.wsrf.TopicListener
- org.globus.wsrf.TopicListenerList

- Less stable API:
  - org.globus.wsrf.NotificationConsumerCallbackManager
  - org.globus.wsrf.NotificationConsumerManager
  - org.globus.wsrf.NotifyCallback
  - org.globus.wsrf.encoding.ObjectSerializer
  - org.globus.wsrf.encoding.ObjectDeserializer
  - org.globus.wsrf.impl.SimpleResourceKey
  - org.globus.wsrf.impl.BaseResourceProperty
  - org.globus.wsrf.impl.ReflectionResourceProperty
  - org.globus.wsrf.impl.SimpleResourceProperty
  - org.globus.wsrf.impl.SimpleResourcePropertyMetaData
  - org.globus.wsrf.impl.SimpleResourcePropertySet
  - org.globus.wsrf.impl.ResourceContextImpl
  - org.globus.wsrf.impl.ResourceHomeImpl
  - org.globus.wsrf.impl.SingletonResourceHome
  - org.globus.wsrf.impl.ServiceResourceHome
  - org.globus.wsrf.impl.ResourcePropertyTopic
  - org.globus.wsrf.impl.SimpleTopic
  - org.globus.wsrf.impl.SimpleTopicList
  - org.globus.wsrf.impl.SimpleTopicListMetaData
  - org.globus.wsrf.query.QueryEngine
  - org.globus.wsrf.query.ExpressionEvaluator
• org.globus.wsrf.topicexpression.TopicExpressionEngine
• org.globus.wsrf.topicexpression.TopicExpressionEvaluator
• org.globus.wsrf.utils.FaultHelper
• org.globus.wsrf.utils.XmlUtils
• org.globus.wsrf.utils.AddressingUtils
• org.globus.wsrf.container.ServiceHost

The complete Java WS Core API\(^1\).

\(^1\) http://www.mcs.anl.gov/~gawor/javawscore/HEAD/doc/javadocs/
Chapter 7. Services and WSDL

Java WS Core contains an implementation of the following specifications. Please see the corresponding specifications for details:

- **WSRF**\(^1\)
  - WS-Resource (WSRF-Resource) specification\(^2\) [wsdl\(^3\) | xsd\(^4\)]
  - WS-ResourceProperties (WSRF-RP) specification\(^5\) [wsdl\(^6\) | xsd\(^7\)]
  - WS-ResourceLifetime (WSRF-RL) specification\(^8\) [wsdl\(^9\) | xsd\(^10\)]
  - WS-ServiceGroup (WSRF-SG) specification\(^11\) [wsdl\(^12\) | xsd\(^13\)]
  - WS-BaseFaults (WSRF-BF) specification\(^14\) [wsdl\(^15\) | xsd\(^16\)]
- **WSN**\(^17\)
  - WS-BaseNotification specification\(^18\) [wsdl\(^19\) | xsd\(^20\)]
  - WS-Topics specification\(^21\) [xsd\(^22\)]
- **WSA**\(^23\)
  - WSA Core\(^24\) [xsd\(^25\)]
  - WSA SOAP Binding\(^26\) [xsd\(^27\)]

---

11. http://docs.oasis-open.org/wsrfrf-ws_service_group-1.2-spec-os.pdf
23. http://docs.oasis-open.org/wsrfrf-ws_topics-1.3-spec-os.pdf
Java WS Core Commands

These command line tools are available on Unix and Windows platforms and will work in the same way (of course within the platform rules - the path syntax, variable definitions, etc.).

The wsrf-* and wsn-* clients should work against any service that supports the given WSRF or WSN operations.
Name

globus-start-container -- Starts standalone container

globus-start-container

Tool description

Starts a standalone container. By default a secure container is started on port 8443 and is accessible via HTTPS. On successful startup a list of services will be displayed on the console. By default the non secure (HTTP) container is started on port 8080.

Command syntax

globus-start-container [options]

Table 3. Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-help</td>
<td>Displays help information about the command.</td>
</tr>
<tr>
<td>-p &lt;port&gt;</td>
<td>Sets the port number for the container.</td>
</tr>
<tr>
<td>-i &lt;address&gt;</td>
<td>Binds container to the specified network address.</td>
</tr>
<tr>
<td>-quiet</td>
<td>Does not show a list of services at startup.</td>
</tr>
<tr>
<td>-debug</td>
<td>Enables debug mode.</td>
</tr>
<tr>
<td>-nosec</td>
<td>Starts a non secure (HTTP) container. Please note that this option only</td>
</tr>
<tr>
<td></td>
<td>disables transport security. Message security can still be used.</td>
</tr>
<tr>
<td>-containerDesc &lt;file&gt;</td>
<td>Specifies a container security descriptor file.</td>
</tr>
<tr>
<td>-profile &lt;name&gt;</td>
<td>Specifies a configuration profile name for the container.</td>
</tr>
</tbody>
</table>
Name

globus-stop-container -- Stops standalone container

globus-stop-container

Tool description

Stops a standalone container. By default this command will attempt to stop a container running on localhost:8443 and perform a soft shutdown.

The globus-stop-container command invokes a ShutdownService running in the container. By default that service is configured to perform self authorization and therefore the globus-stop-container must be executed with the same credentials as the container it is running with. Alternatively, the service can be configured with a gridmap file to allow a subset of users (with their own credentials) to invoke the service (please see the service security deployment descriptor for details).

Command syntax

globus-stop-container [options] ['soft' | 'hard']
Table 4. Common options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Displays help information about the command.</td>
</tr>
<tr>
<td>-d, --debug</td>
<td>Enables debug mode. For example, full stack traces of errors will be displayed.</td>
</tr>
<tr>
<td>-e, --eprFile &lt;file&gt;</td>
<td>Specifies an XML file that contains the WS-Addressing endpoint reference.</td>
</tr>
<tr>
<td>-s, --service &lt;url&gt;</td>
<td>Specifies the service URL.</td>
</tr>
</tbody>
</table>
| -k, --key <name value> | Specifies the resource key. The name is the QName of the resource key in the string form: {namespaceURI}localPart, while the value is the simple value of the key. For complex keys, use the --eprFile option. Example:  
- k "(http://www.globus.org)MyKey" |
| -f, --descriptor <file> | Specifies a client security descriptor. Overrides all other security settings. |
| -a, --anonymous | Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism. |
| -g, --delegation <mode> | Enables delegation. mode can be either 'limited' or 'full'. Only supported with the GSI Secure Conversation authentication mechanism. |
| -l, --contextLifetime <value> | Sets the lifetime of the client security context. value is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism. |
| -m, --securityMech <type> | Specifies the authentication mechanism. type can be 'msg' for GSI Secure Message, or 'conv' for GSI Secure Conversation. |
| -c, --serverCertificate <file> | Specifies the server’s certificate file used for encryption. Only needed for the GSI Secure Message authentication mechanism. |
| -p, --protection <type> | Specifies the protection level. type can be 'sig' for signature or 'enc' for encryption. |
| -x, --proxyFilename <value> | Sets the proxy file to use as client credential. |
| -z, --authorization <type> | Specifies authorization type. type can be 'self', 'host', 'none', or a string specifying the expected identity of the remote party. |
| -t, --timeout <timeout> | Specifies client timeout (in seconds). The client will wait maximum of the timeout value for a response from the server before returning an error. By default the timeout value is 10 minutes. |

Table 5. Shutdown options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'soft'</td>
<td>This option lets the threads die naturally.</td>
</tr>
<tr>
<td>'hard'</td>
<td>This option forces an immediate JVM shutdown.</td>
</tr>
</tbody>
</table>

Example:

$ globus-stop-container soft

Please see the troubleshooting section if you are having problems with globus-stop-container.
Name

globus-start-container-detached -- Starts standalone container detached from controlling TTY

globus-start-container-detached

Tool description

Starts a standalone container detached from the controlling TTY. This can be useful for long running containers or when started from init.d scripts. Container log goes to $GLOBUS_LOCATION/var/container.log and a PID file is written to $GLOBUS_LOCATION/var/container.pid. globus-start-container-detached is just a wrapper around globus-start-container so see globus-start-container for more information and options.

Note

Note that this tool is only available after doing a full Globus install. It is not available in Java WS Core only installs.

Command syntax

globus-start-container-detached [options] | [arguments to container]

Table 6. Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-logfile &lt;file&gt;</td>
<td>Specifies an alternate container log file.</td>
</tr>
<tr>
<td>-append</td>
<td>Do not overwrite the existing log file.</td>
</tr>
<tr>
<td>-pidfile &lt;file&gt;</td>
<td>Specifies an alternate PID file location.</td>
</tr>
</tbody>
</table>
**Name**

globus-stop-container-detached -- Stops standalone container detached from controlling TTY

globus-stop-container-detached

**Tool description**

Stops a standalone container detached from the controlling TTY. $GLOBUS_LOCATION/var/container.pid is used to find the PID of the running container and signals are sent to stop the container.

⚠️ **Note**

Note that this tool is only available after doing a full Globus install. It is not available in Java WS Core only installs.

**Command syntax**

    globus-stop-container-detached [options]

**Table 7. Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-pidfile &lt;file&gt;</td>
<td>Specifies an alternate PID file location.</td>
</tr>
</tbody>
</table>
Name

wsrf-destroy -- Destroys a resource

wsrf-destroy

Tool description

Destroys a resource.

Command syntax

wsrf-destroy [options]
Table 8. Common options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-h, --help</code></td>
<td>Displays help information about the command.</td>
</tr>
<tr>
<td><code>-d, --debug</code></td>
<td>Enables debug mode. For example, full stack traces of errors will be displayed.</td>
</tr>
<tr>
<td><code>-e, --eprFile &lt;file&gt;</code></td>
<td>Specifies an XML file that contains the WS-Addressing endpoint reference.</td>
</tr>
<tr>
<td><code>-s, --service &lt;url&gt;</code></td>
<td>Specifies the service URL.</td>
</tr>
<tr>
<td><code>-k, --key &lt;name value&gt;</code></td>
<td>Specifies the resource key. The name is the QName of the resource key in the string form: {namespaceURI}localPart, while the value is the simple value of the key. For complex keys, use the --eprFile option. Example: -k &quot;(<a href="http://www.globus.org)MyKey">http://www.globus.org)MyKey</a>&quot; 123</td>
</tr>
<tr>
<td><code>-f, --descriptor &lt;file&gt;</code></td>
<td>Specifies a client security descriptor. Overrides all other security settings.</td>
</tr>
<tr>
<td><code>-a, --anonymous</code></td>
<td>Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td><code>-g, --delegation &lt;mode&gt;</code></td>
<td>Enables delegation. mode can be either 'limited' or 'full'. Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td><code>-l, --contextLifetime &lt;value&gt;</code></td>
<td>Sets the lifetime of the client security context. value is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td><code>-m, --securityMech &lt;type&gt;</code></td>
<td>Specifies the authentication mechanism. type can be 'msg' for GSI Secure Message, or 'conv' for GSI Secure Conversation.</td>
</tr>
<tr>
<td><code>-c, --serverCertificate &lt;file&gt;</code></td>
<td>Specifies the server’s certificate file used for encryption. Only needed for the GSI Secure Message authentication mechanism.</td>
</tr>
<tr>
<td><code>-p, --protection &lt;type&gt;</code></td>
<td>Specifies the protection level. type can be 'sig' for signature or 'enc' for encryption.</td>
</tr>
<tr>
<td><code>-x, --proxyFilename &lt;value&gt;</code></td>
<td>Sets the proxy file to use as client credential.</td>
</tr>
<tr>
<td><code>-z, --authorization &lt;type&gt;</code></td>
<td>Specifies authorization type. type can be 'self', 'host', 'none', or a string specifying the expected identity of the remote party.</td>
</tr>
<tr>
<td><code>-t, --timeout &lt;timeout&gt;</code></td>
<td>Specifies client timeout (in seconds). The client will wait maximum of the timeout value for a response from the server before returning an error. By default the timeout value is 10 minutes.</td>
</tr>
</tbody>
</table>

Example:

```
$ wsrfs-destroy -s http://localhost:8080/wsrfs/services/CounterService \ -k "(http://counter.com)CounterKey" 123
```
Name

wsrf-set-termination-time -- Sets termination time of a resource

wsrf-set-termination-time

Tool description

Sets a termination time of a resource.

Command syntax

wsrf-set-termination-time [options] <seconds> | 'infinity'

The following are command-specific options in addition to the common options:

Table 9. Command-specific options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-u, --utc</td>
<td>Display time in UTC.</td>
</tr>
</tbody>
</table>
### Table 10. Common options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-h, --help</code></td>
<td>Displays help information about the command.</td>
</tr>
<tr>
<td><code>-d, --debug</code></td>
<td>Enables debug mode. For example, full stack traces of errors will be displayed.</td>
</tr>
<tr>
<td><code>-e, --eprFile &lt;file&gt;</code></td>
<td>Specifies an XML file that contains the WS-Addressing endpoint reference.</td>
</tr>
<tr>
<td><code>-s, --service &lt;url&gt;</code></td>
<td>Specifies the service URL.</td>
</tr>
<tr>
<td><code>-k, --key &lt;name value&gt;</code></td>
<td>Specifies the resource key. The name is the QName of the resource key in the string form: <code>{namespaceURI}localPart</code>, while the value is the simple value of the key. For complex keys, use the --eprFile option. Example: <code>-k &quot;{http://www.globus.org}MyKey&quot; 123</code></td>
</tr>
<tr>
<td><code>-f, --descriptor &lt;file&gt;</code></td>
<td>Specifies a client security descriptor.overrides all other security settings.</td>
</tr>
<tr>
<td><code>-a, --anonymous</code></td>
<td>Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td><code>-g, --delegation &lt;mode&gt;</code></td>
<td>Enables delegation. mode can be either 'limited' or 'full'. Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td><code>-l, --contextLifetime &lt;value&gt;</code></td>
<td>Sets the lifetime of the client security context. value is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td><code>-m, --securityMech &lt;type&gt;</code></td>
<td>Specifies the authentication mechanism. type can be 'msg' for GSI Secure Message, or 'conv' for GSI Secure Conversation.</td>
</tr>
<tr>
<td><code>-c, --serverCertificate &lt;file&gt;</code></td>
<td>Specifies the server's certificate file used for encryption. Only needed for the GSI Secure Message authentication mechanism.</td>
</tr>
<tr>
<td><code>-p, --protection &lt;type&gt;</code></td>
<td>Specifies the protection level. type can be 'sig' for signature or 'enc' for encryption.</td>
</tr>
<tr>
<td><code>-x, --proxyFilename &lt;value&gt;</code></td>
<td>Sets the proxy file to use as client credential.</td>
</tr>
<tr>
<td><code>-z, --authorization &lt;type&gt;</code></td>
<td>Specifies authorization type. type can be 'self', 'host', 'none', or a string specifying the expected identity of the remote party.</td>
</tr>
<tr>
<td><code>-t, --timeout &lt;timeout&gt;</code></td>
<td>Specifies client timeout (in seconds). The client will wait maximum of the timeout value for a response from the server before returning an error. By default the timeout value is 10 minutes.</td>
</tr>
</tbody>
</table>

Example:

```bash
$ wsrf-set-termination-time -s http://localhost:8080/wsrfs/services/CounterService -k "{http://counter.com}CounterKey" 123 30
```
**Name**

wsrf-query -- Performs query on a resource property document

wsrf-query

**Tool description**

Queries the resource property document of a resource. By default, a simple XPath query is assumed that returns the entire resource property document.

**Command syntax**

wsrf-query [options] [query expression] [dialect]
Table 11. Common options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-h, --help</code></td>
<td>Displays help information about the command.</td>
</tr>
<tr>
<td><code>-d, --debug</code></td>
<td>Enables debug mode. For example, full stack traces of errors will be displayed.</td>
</tr>
<tr>
<td><code>-e, --eprFile &lt;file&gt;</code></td>
<td>Specifies an XML file that contains the WS-Addressing endpoint reference.</td>
</tr>
<tr>
<td><code>-s, --service &lt;url&gt;</code></td>
<td>Specifies the service URL.</td>
</tr>
<tr>
<td><code>-k, --key &lt;name value&gt;</code></td>
<td>Specifies the resource key. The name is the QName of the resource key in the string form: [namespaceURI]localPart, while the value is the simple value of the key. For complex keys, use the --eprFile option. Example: <code>-k &quot;(http://www.globus.org)MyKey&quot; 123</code></td>
</tr>
<tr>
<td><code>-f, --descriptor &lt;file&gt;</code></td>
<td>Specifies a client security descriptor. Overrides all other security settings.</td>
</tr>
<tr>
<td><code>-a, --anonymous</code></td>
<td>Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td><code>-g, --delegation &lt;mode&gt;</code></td>
<td>Enables delegation. mode can be either 'limited' or 'full'. Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td><code>-l, --contextLifetime &lt;value&gt;</code></td>
<td>Sets the lifetime of the client security context. value is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td><code>-m, --securityMech &lt;type&gt;</code></td>
<td>Specifies the authentication mechanism. type can be 'msg' for GSI Secure Message, or 'conv' for GSI Secure Conversation.</td>
</tr>
<tr>
<td><code>-c, --serverCertificate &lt;file&gt;</code></td>
<td>Specifies the server's certificate file used for encryption. Only needed for the GSI Secure Message authentication mechanism.</td>
</tr>
<tr>
<td><code>-p, --protection &lt;type&gt;</code></td>
<td>Specifies the protection level. type can be 'sig' for signature or 'enc' for encryption.</td>
</tr>
<tr>
<td><code>-x, --proxyFilename &lt;value&gt;</code></td>
<td>Sets the proxy file to use as client credential.</td>
</tr>
<tr>
<td><code>-z, --authorization &lt;type&gt;</code></td>
<td>Specifies authorization type. type can be 'self', 'host', 'none', or a string specifying the expected identity of the remote party.</td>
</tr>
<tr>
<td><code>-t, --timeout &lt;timeout&gt;</code></td>
<td>Specifies client timeout (in seconds). The client will wait maximum of the timeout value for a response from the server before returning an error. By default the timeout value is 10 minutes.</td>
</tr>
</tbody>
</table>

Examples:

```
$ wsrf-query -s https://127.0.0.1:8443/wsrfservices/DefaultIndexService "count(//*[local-name()='Entry'])"
```

```
$ wsrf-query -s https://127.0.0.1:8443/wsrfservices/DefaultIndexService "number(//*[local-name()='GLUECE']/glue:ComputingElement/glue:State/@glue:FreeCPUs)=0"
```

```
$ wsrf-query -s http://localhost:8080/wsrfservices/ContainerRegistryService "/*//*//*///*[local-name()='Address']"
```
**Name**

wsrf-get-property -- Gets values of a single resource property

wsrf-get-property

**Tool description**

Gets a single resource property from a resource.

**Command syntax**

wsrf-get-property [options] <property>

The `<property>` is a QName of the resource property in the string form: `{namespaceURI}localPart`.
### Table 12. Common options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-h, --help</code></td>
<td>Displays help information about the command.</td>
</tr>
<tr>
<td><code>-d, --debug</code></td>
<td>Enables debug mode. For example, full stack traces of errors will be displayed.</td>
</tr>
<tr>
<td><code>-e, --eprFile &lt;file&gt;</code></td>
<td>Specifies an XML file that contains the <code>WS-Addressing</code> endpoint reference.</td>
</tr>
<tr>
<td><code>-s, --service &lt;url&gt;</code></td>
<td>Specifies the service URL.</td>
</tr>
<tr>
<td><code>-k, --key &lt;name value&gt;</code></td>
<td>Specifies the resource key. The <code>name</code> is the QName of the resource key in the string form: <code>{namespaceURI}localPart</code>, while the <code>value</code> is the simple value of the key. For complex keys, use the <code>--eprFile</code> option. Example: <code>-k &quot;(http://www.globus.org)MyKey&quot; 123</code></td>
</tr>
<tr>
<td><code>-f, --descriptor &lt;file&gt;</code></td>
<td>Specifies a client security descriptor. Overrides all other security settings.</td>
</tr>
<tr>
<td><code>-a, --anonymous</code></td>
<td>Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td><code>-g, --delegation &lt;mode&gt;</code></td>
<td>Enables delegation. <code>mode</code> can be either 'limited' or 'full'. Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td><code>-l, --contextLifetime &lt;value&gt;</code></td>
<td>Sets the lifetime of the client security context. <code>value</code> is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td><code>-m, --securityMech &lt;type&gt;</code></td>
<td>Specifies the authentication mechanism. <code>type</code> can be 'msg' for GSI Secure Message, or 'conv' for GSI Secure Conversation.</td>
</tr>
<tr>
<td><code>-c, --serverCertificate &lt;file&gt;</code></td>
<td>Specifies the server's certificate file used for encryption. Only needed for the GSI Secure Message authentication mechanism.</td>
</tr>
<tr>
<td><code>-p, --protection &lt;type&gt;</code></td>
<td>Specifies the protection level. <code>type</code> can be 'sig' for signature or 'enc' for encryption.</td>
</tr>
<tr>
<td><code>-x, --proxyFilename &lt;value&gt;</code></td>
<td>Sets the proxy file to use as client credential.</td>
</tr>
<tr>
<td><code>-z, --authorization &lt;type&gt;</code></td>
<td>Specifies authorization type. <code>type</code> can be 'self', 'host', 'none', or a string specifying the expected identity of the remote party.</td>
</tr>
<tr>
<td><code>-t, --timeout &lt;timeout&gt;</code></td>
<td>Specifies client timeout (in seconds). The client will wait maximum of the timeout value for a response from the server before returning an error. By default the timeout value is 10 minutes.</td>
</tr>
</tbody>
</table>

Example:

```bash
```
**Name**

wsrf-get-properties -- Gets values of multiple resource properties

wsrf-get-properties

**Tool description**

Gets multiple resource properties from a resource.

**Command syntax**

```
wsrf-get-properties [options] <property1> [<property2>... <propertyN>]
```

Each `<propertyN>` is a QName of the resource property in the string form: `{namespaceURI}localPart`. 
Table 13. Common options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Displays help information about the command.</td>
</tr>
<tr>
<td>-d, --debug</td>
<td>Enables debug mode. For example, full stack traces of errors will be dis-</td>
</tr>
<tr>
<td></td>
<td>played.</td>
</tr>
<tr>
<td>-e, --eprFile &lt;file&gt;</td>
<td>Specifies an XML file that contains the WS-Addressing endpoint reference.</td>
</tr>
<tr>
<td>-s, --service &lt;url&gt;</td>
<td>Specifies the service URL.</td>
</tr>
<tr>
<td>-k, --key &lt;name value&gt;</td>
<td>Specifies the resource key. The name is the QName of the resource key in</td>
</tr>
<tr>
<td></td>
<td>the string form: {namespaceURI}localPart, while the value is the simple</td>
</tr>
<tr>
<td></td>
<td>value of the key. For complex keys, use the --eprFile option. Example:</td>
</tr>
<tr>
<td></td>
<td>-k &quot;{<a href="http://www.globus.org%5C%7DMyKey">http://www.globus.org\}MyKey</a>&quot; 123</td>
</tr>
<tr>
<td>-f, --descriptor &lt;file&gt;</td>
<td>Specifies a client security descriptor. Overrides all other security</td>
</tr>
<tr>
<td></td>
<td>settings.</td>
</tr>
<tr>
<td>-a, --anonymous</td>
<td>Enables anonymous authentication. Only supported with transport security</td>
</tr>
<tr>
<td></td>
<td>or the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-g, --delegation &lt;mode&gt;</td>
<td>Enables delegation. mode can be either 'limited' or 'full'. Only supported</td>
</tr>
<tr>
<td></td>
<td>with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-l, --contextLifetime &lt;value&gt;</td>
<td>Sets the lifetime of the client security context. value is in milliseconds.</td>
</tr>
<tr>
<td></td>
<td>Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-m, --securityMech &lt;type&gt;</td>
<td>Specifies the authentication mechanism. type can be 'msg' for GSI Secure</td>
</tr>
<tr>
<td></td>
<td>Message, or 'conv' for GSI Secure Conversation.</td>
</tr>
<tr>
<td>-c, --serverCertificate &lt;file&gt;</td>
<td>Specifies the server's certificate file used for encryption. Only needed</td>
</tr>
<tr>
<td></td>
<td>for the GSI Secure Message authentication mechanism.</td>
</tr>
<tr>
<td>-p, --protection &lt;type&gt;</td>
<td>Specifies the protection level. type can be 'sig' for signature or 'enc'</td>
</tr>
<tr>
<td></td>
<td>for encryption.</td>
</tr>
<tr>
<td>-x, --proxyFilename &lt;value&gt;</td>
<td>Sets the proxy file to use as client credential.</td>
</tr>
<tr>
<td>-z, --authorization &lt;type&gt;</td>
<td>Specifies authorization type. type can be 'self', 'host', 'none', or a</td>
</tr>
<tr>
<td></td>
<td>string specifying the expected identity of the remote party.</td>
</tr>
<tr>
<td>-t, --timeout &lt;timeout&gt;</td>
<td>Specifies client timeout (in seconds). The client will wait maximum of</td>
</tr>
<tr>
<td></td>
<td>the timeout value for a response from the server before returning an error.</td>
</tr>
<tr>
<td></td>
<td>By default the timeout value is 10 minutes.</td>
</tr>
</tbody>
</table>

Example:

```
$ wsrf-get-properties \-s http://localhost:8080/wsrfservices/CounterService \ 
"\{http://counter.com\}CounterKey" 123 \ 
"\{http://docs.oasis-open.org/wsrfs/2004/06/wsrfs-WS-ResourceLifetime-1.2-draft-01.xsd\}Cu \ 
"\{http://docs.oasis-open.org/wsrfs/2004/06/wsrfs-WS-ResourceLifetime-1.2-draft-01.xsd\}Te
```
Name

wsrf-insert-property -- Inserts values into a resource property

wsrf-insert-property

Tool description

Inserts a property into the resource property document of a resource.

Command syntax

wsrf-insert-property [options] <propertyValueFile>

The <propertyValueFile> is an XML file that contains the value of the resource property. The QName of the property is the outer most element.
Table 14. Common options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Displays help information about the command.</td>
</tr>
<tr>
<td>-d, --debug</td>
<td>Enables debug mode. For example, full stack traces of errors will be displayed.</td>
</tr>
<tr>
<td>-e, --eprFile &lt;file&gt;</td>
<td>Specifies an XML file that contains the WS-Addressing endpoint reference.</td>
</tr>
<tr>
<td>-s, --service &lt;url&gt;</td>
<td>Specifies the service URL.</td>
</tr>
<tr>
<td>-k, --key &lt;name value&gt;</td>
<td>Specifies the resource key. The name is the QName of the resource key in the string form: <code>{namespaceURI}localPart</code>, while the value is the simple value of the key. For complex keys, use the --eprFile option. Example: <code>-k &quot;(http://www.globus.org)MyKey&quot; 123</code></td>
</tr>
<tr>
<td>-f, --descriptor &lt;file&gt;</td>
<td>Specifies a client security descriptor. Overrides all other security settings.</td>
</tr>
<tr>
<td>-a, --anonymous</td>
<td>Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-g, --delegation &lt;mode&gt;</td>
<td>Enables delegation. mode can be either 'limited' or 'full'. Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-l, --contextLifetime &lt;value&gt;</td>
<td>Sets the lifetime of the client security context. value is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-m, --securityMech &lt;type&gt;</td>
<td>Specifies the authentication mechanism. type can be 'msg' for GSI Secure Message, or 'conv' for GSI Secure Conversation.</td>
</tr>
<tr>
<td>-c, --serverCertificate &lt;file&gt;</td>
<td>Specifies the server’s certificate file used for encryption. Only needed for the GSI Secure Message authentication mechanism.</td>
</tr>
<tr>
<td>-p, --protection &lt;type&gt;</td>
<td>Specifies the protection level. type can be 'sig' for signature or 'enc' for encryption.</td>
</tr>
<tr>
<td>-x, --proxyFilename &lt;value&gt;</td>
<td>Sets the proxy file to use as client credential.</td>
</tr>
<tr>
<td>-z, --authorization &lt;type&gt;</td>
<td>Specifies authorization type. type can be 'self', 'host', 'none', or a string specifying the expected identity of the remote party.</td>
</tr>
<tr>
<td>-t, --timeout &lt;timeout&gt;</td>
<td>Specifies client timeout (in seconds). The client will wait maximum of the timeout value for a response from the server before returning an error. By default the timeout value is 10 minutes.</td>
</tr>
</tbody>
</table>

Example: Contents of in.xml:

```
<doc> <ns1:foo xmlns:ns1="http://widgets.com"> Value1 
  </ns1:foo> <ns1:foo xmlns:ns1="http://widgets.com"> Value2 
</ns1:foo> </doc>
```

```
```
Name

wsrf-update-property -- Updates value of a resource property

wsrf-update-property

Tool description

Updates the property value in the resource property document of a resource.

Command syntax

wsrf-update-property [options] <propertyValueFile>

The <propertyValueFile> is an XML file that contains the value of the resource property. The QName of the property is the outermost element.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Displays help information about the command.</td>
</tr>
<tr>
<td>-d, --debug</td>
<td>Enables debug mode. For example, full stack traces of errors will be displayed.</td>
</tr>
<tr>
<td>-e, --eprFile &lt;file&gt;</td>
<td>Specifies an XML file that contains the WS-Addressing endpoint reference.</td>
</tr>
<tr>
<td>-s, --service &lt;url&gt;</td>
<td>Specifies the service URL.</td>
</tr>
<tr>
<td>-k, --key &lt;name value&gt;</td>
<td>Specifies the resource key. The name is the QName of the resource key in the string form: {namespaceURI}localPart, while the value is the simple value of the key. For complex keys, use the --eprFile option. Example: -k &quot;(<a href="http://www.globus.org)MyKey">http://www.globus.org)MyKey</a>&quot; 123</td>
</tr>
<tr>
<td>-f, --descriptor &lt;file&gt;</td>
<td>Specifies a client security descriptor. Overrides all other security settings.</td>
</tr>
<tr>
<td>-a, --anonymous</td>
<td>Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-g, --delegation &lt;mode&gt;</td>
<td>Enables delegation. mode can be either 'limited' or 'full'. Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-l, --contextLifetime &lt;value&gt;</td>
<td>Sets the lifetime of the client security context. value is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-m, --securityMech &lt;type&gt;</td>
<td>Specifies the authentication mechanism. type can be 'msg' for GSI Secure Message, or 'conv' for GSI Secure Conversation.</td>
</tr>
<tr>
<td>-c, --serverCertificate &lt;file&gt;</td>
<td>Specifies the server’s certificate file used for encryption. Only needed for the GSI Secure Message authentication mechanism.</td>
</tr>
<tr>
<td>-p, --protection &lt;type&gt;</td>
<td>Specifies the protection level. type can be 'sig' for signature or 'enc' for encryption.</td>
</tr>
<tr>
<td>-x, --proxyFilename &lt;value&gt;</td>
<td>Sets the proxy file to use as client credential.</td>
</tr>
<tr>
<td>-z, --authorization &lt;type&gt;</td>
<td>Specifies authorization type. type can be 'self', 'host', 'none', or a string specifying the expected identity of the remote party.</td>
</tr>
<tr>
<td>-t, --timeout &lt;timeout&gt;</td>
<td>Specifies client timeout (in seconds). The client will wait maximum of the timeout value for a response from the server before returning an error. By default the timeout value is 10 minutes.</td>
</tr>
</tbody>
</table>

Example: Contents of in.xml:

```xml
<doc> <ns1:foo xmlns:ns1="http://widgets.com"> Value </ns1:foo> </doc>
```

```
```
Name

wsrf-delete-property -- Deletes a resource property

wsrf-delete-property

Tool description

Deletes a resource property from the resource property document of a resource.

Command syntax

wsrf-delete-property [options] <property>

The <property> is a QName of the resource property in the string form: {namespaceURI}localPart.
Table 16. Common options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Displays help information about the command.</td>
</tr>
<tr>
<td>-d, --debug</td>
<td>Enables debug mode. For example, full stack traces of errors will be displayed.</td>
</tr>
<tr>
<td>-e, --eprFile &lt;file&gt;</td>
<td>Specifies an XML file that contains the WS-Addressing endpoint reference.</td>
</tr>
<tr>
<td>-s, --service &lt;url&gt;</td>
<td>Specifies the service URL.</td>
</tr>
</tbody>
</table>
| -k, --key <name value> | Specifies the resource key. The name is the QName of the resource key in the string form: {namespaceURI}localPart, while the value is the simple value of the key. For complex keys, use the --eprFile option. Example: 
  -k "({http://www.globus.org}MyKey" 123 |
| -f, --descriptor <file> | Specifies a client security descriptor. Overrides all other security settings. |
| -a, --anonymous | Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism. |
| -g, --delegation <mode> | Enables delegation. mode can be either 'limited' or 'full'. Only supported with the GSI Secure Conversation authentication mechanism. |
| -l, --contextLifetime <value> | Sets the lifetime of the client security context. value is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism. |
| -m, --securityMech <type> | Specifies the authentication mechanism. type can be 'msg' for GSI Secure Message, or 'conv' for GSI Secure Conversation. |
| -c, --serverCertificate <file> | Specifies the server's certificate file used for encryption. Only needed for the GSI Secure Message authentication mechanism. |
| -p, --protection <type> | Specifies the protection level. type can be 'sig' for signature or 'enc' for encryption. |
| -x, --proxyFilename <value> | Sets the proxy file to use as client credential. |
| -z, --authorization <type> | Specifies authorization type. type can be 'self', 'host', 'none', or a string specifying the expected identity of the remote party. |
| -t, --timeout <timeout> | Specifies client timeout (in seconds). The client will wait maximum of the timeout value for a response from the server before returning an error. By default the timeout value is 10 minutes. |

Example:

```bash
```
Name

wsn-get-current-message -- Gets a current message associated with a topic

wsn-get-current-message

Tool description

Gets the current message associated with the specified topic.

Command syntax

wsn-get-current-message [options] <topic>

The <topic> is a QName of the resource property in the string form: [namespaceURI]localPart.
Table 17. Common options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Displays help information about the command.</td>
</tr>
<tr>
<td>-d, --debug</td>
<td>Enables debug mode. For example, full stack traces of errors will be displayed.</td>
</tr>
<tr>
<td>-e, --eprFile &lt;file&gt;</td>
<td>Specifies an XML file that contains the WS-Addressing endpoint reference.</td>
</tr>
<tr>
<td>-s, --service &lt;url&gt;</td>
<td>Specifies the service URL.</td>
</tr>
</tbody>
</table>
| -k, --key <name value> | Specifies the resource key. The name is the QName of the resource key in the string form: [namespaceURI]localPart, while the value is the simple value of the key. For complex keys, use the --eprFile option. Example: 

```
-k "http://www.globus.org)MyKey"
123
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-f, --descriptor &lt;file&gt;</td>
<td>Specifies a client security descriptor. Overrides all other security settings.</td>
</tr>
<tr>
<td>-a, --anonymous</td>
<td>Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-g, --delegation &lt;mode&gt;</td>
<td>Enables delegation. mode can be either 'limited' or 'full'. Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-l, --contextLifetime &lt;value&gt;</td>
<td>Sets the lifetime of the client security context. value is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-m, --securityMech &lt;type&gt;</td>
<td>Specifies the authentication mechanism. type can be 'msg' for GSI Secure Message, or 'conv' for GSI Secure Conversation.</td>
</tr>
<tr>
<td>-c, --serverCertificate &lt;file&gt;</td>
<td>Specifies the server's certificate file used for encryption. Only needed for the GSI Secure Message authentication mechanism.</td>
</tr>
<tr>
<td>-p, --protection &lt;type&gt;</td>
<td>Specifies the protection level. type can be 'sig' for signature or 'enc' for encryption.</td>
</tr>
<tr>
<td>-x, --proxyFilename &lt;value&gt;</td>
<td>Sets the proxy file to use as client credential.</td>
</tr>
<tr>
<td>-z, --authorization &lt;type&gt;</td>
<td>Specifies authorization type. type can be 'self', 'host', 'none', or a string specifying the expected identity of the remote party.</td>
</tr>
<tr>
<td>-t, --timeout &lt;timeout&gt;</td>
<td>Specifies client timeout (in seconds). The client will wait maximum of the timeout value for a response from the server before returning an error. By default the timeout value is 10 minutes.</td>
</tr>
</tbody>
</table>

Example:

```
```
Name

wsn-pause-subscription -- Pauses a subscription

wsn-pause-subscription

Tool description

Pauses a subscription (notifications on that subscription will not be sent out until it is resumed).

Command syntax

wsn-pause-subscription [options]
Table 18. Common options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Displays help information about the command.</td>
</tr>
<tr>
<td>-d, --debug</td>
<td>Enables debug mode. For example, full stack traces of errors will be displayed.</td>
</tr>
<tr>
<td>-e, --eprFile &lt;file&gt;</td>
<td>Specifies an XML file that contains the WS-Addressing endpoint reference.</td>
</tr>
<tr>
<td>-s, --service &lt;url&gt;</td>
<td>Specifies the service URL.</td>
</tr>
</tbody>
</table>
| -k, --key <name value> | Specifies the resource key. The name is the QName of the resource key in the string form: \{namespaceURI\}localPart, while the value is the simple value of the key. For complex keys, use the --eprFile option. Example: 
-k "\{http://www.globus.org\}MyKey" |
| -f, --descriptor <file> | Specifies a client security descriptor. Overrides all other security settings. |
| -a, --anonymous | Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism. |
| -g, --delegation <mode> | Enables delegation. mode can be either 'limited' or 'full'. Only supported with the GSI Secure Conversation authentication mechanism. |
| -l, --contextLifetime <value> | Sets the lifetime of the client security context. value is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism. |
| -m, --securityMech <type> | Specifies the authentication mechanism. type can be 'msg' for GSI Secure Message, or 'conv' for GSI Secure Conversation. |
| -c, --serverCertificate <file> | Specifies the server's certificate file used for encryption. Only needed for the GSI Secure Message authentication mechanism. |
| -p, --protection <type> | Specifies the protection level. type can be 'sig' for signature or 'enc' for encryption. |
| -x, --proxyFilename <value> | Sets the proxy file to use as client credential. |
| -z, --authorization <type> | Specifies authorization type. type can be 'self', 'host', 'none', or a string specifying the expected identity of the remote party. |
| -t, --timeout <timeout> | Specifies client timeout (in seconds). The client will wait maximum of the timeout value for a response from the server before returning an error. By default the timeout value is 10 minutes. |

Example:

```bash
$ wsn-pause-subscription -s http://localhost:8080/wsrf/services/SubscriptionManagerService \ -k "(http://www.globus.org/namespaces/2004/06/core)acc271c0-4df9-11d9-ab19-87da3bc7cf28"
```
Name

wsn-resume-subscription -- Resumes a subscription

wsn-resume-subscription

Tool description

Resumes a subscription (notifications on that subscription will be sent out again).

Command syntax

wsn-resume-subscription [options]
Table 19. Common options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Displays help information about the command.</td>
</tr>
<tr>
<td>-d, --debug</td>
<td>Enables debug mode. For example, full stack traces of errors will be displayed.</td>
</tr>
<tr>
<td>-e, --eprFile &lt;file&gt;</td>
<td>Specifies an XML file that contains the WS-Addressing endpoint reference.</td>
</tr>
<tr>
<td>-s, --service &lt;url&gt;</td>
<td>Specifies the service URL.</td>
</tr>
</tbody>
</table>
| -k, --key <name value> | Specifies the resource key. The name is the QName of the resource key in the string form: {namespaceURI}localPart, while the value is the simple value of the key. For complex keys, use the --eprFile option. Example: 
  \[-k "(http://www.globus.org)MyKey"
  123\] |
| -f, --descriptor <file> | Specifies a client security descriptor. Overrides all other security settings. |
| -a, --anonymous | Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism. |
| -g, --delegation <mode> | Enables delegation. mode can be either 'limited' or 'full'. Only supported with the GSI Secure Conversation authentication mechanism. |
| -l, --contextLifetime <value> | Sets the lifetime of the client security context. value is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism. |
| -m, --securityMech <type> | Specifies the authentication mechanism. type can be 'msg' for GSI Secure Message, or 'conv' for GSI Secure Conversation. |
| -c, --serverCertificate <file> | Specifies the server's certificate file used for encryption. Only needed for the GSI Secure Message authentication mechanism. |
| -p, --protection <type> | Specifies the protection level. type can be 'sig' for signature or 'enc' for encryption. |
| -x, --proxyFilename <value> | Sets the proxy file to use as client credential.                           |
| -z, --authorization <type> | Specifies authorization type. type can be 'self', 'host', 'none', or a string specifying the expected identity of the remote party. |
| -t, --timeout <timeout> | Specifies client timeout (in seconds). The client will wait maximum of the timeout value for a response from the server before returning an error. By default the timeout value is 10 minutes. |

Example:

```
$ wsn-resume-subscription -s http://localhost:8080/wsrf/services/SubscriptionManagerService \ -k "(http://www.globus.org/namespaces/2004/06/core)acc271c0-4df9-11d9-ab19-87da3bc7cf28"
```
Name

wsn-subscribe -- Subscribes to a topic

wsn-subscribe

Tool description

Subscribes to a topic.

Command syntax

wsn-subscribe [options] <topic>

The <topic> is a QName of the resource property in the string form: {namespaceURI}localPart.

The following are subscribe-specific options in addition to the common options:

Table 20. Command-specific options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-r, --resDescFile &lt;file&gt;</td>
<td>Specifies a file containing a resource security descriptor for the notification consumer resource.</td>
</tr>
<tr>
<td>-b, --subEpr &lt;file&gt;</td>
<td>Specifies a file to which the subscription resource EPR will be saved.</td>
</tr>
</tbody>
</table>
### Table 21. Common options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-h, --help</code></td>
<td>Displays help information about the command.</td>
</tr>
<tr>
<td><code>-d, --debug</code></td>
<td>Enables debug mode. For example, full stack traces of errors will be displayed.</td>
</tr>
<tr>
<td><code>-e, --eprFile &lt;file&gt;</code></td>
<td>Specifies an XML file that contains the WS-Addressing endpoint reference.</td>
</tr>
<tr>
<td><code>-s, --service &lt;url&gt;</code></td>
<td>Specifies the service URL.</td>
</tr>
<tr>
<td><code>-k, --key &lt;name value&gt;</code></td>
<td>Specifies the resource key. The name is the QName of the resource key in the string form: <code>{namespaceURI}localPart</code>, while the value is the simple value of the key. For complex keys, use the --eprFile option. Example: <code>-k &quot;({http://www.globus.org}MyKey&quot; 123</code></td>
</tr>
<tr>
<td><code>-f, --descriptor &lt;file&gt;</code></td>
<td>Specifies a client security descriptor. Overrides all other security settings.</td>
</tr>
<tr>
<td><code>-a, --anonymous</code></td>
<td>Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td><code>-g, --delegation &lt;mode&gt;</code></td>
<td>Enables delegation. mode can be either 'limited' or 'full'. Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td><code>-l, --contextLifetime &lt;value&gt;</code></td>
<td>Sets the lifetime of the client security context. value is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td><code>-m, --securityMech &lt;type&gt;</code></td>
<td>Specifies the authentication mechanism. type can be 'msg' for GSI Secure Message, or 'conv' for GSI Secure Conversation.</td>
</tr>
<tr>
<td><code>-c, --serverCertificate &lt;file&gt;</code></td>
<td>Specifies the server’s certificate file used for encryption. Only needed for the GSI Secure Message authentication mechanism.</td>
</tr>
<tr>
<td><code>-p, --protection &lt;type&gt;</code></td>
<td>Specifies the protection level. type can be 'sig' for signature or 'enc' for encryption.</td>
</tr>
<tr>
<td><code>-x, --proxyFilename &lt;value&gt;</code></td>
<td>Sets the proxy file to use as client credential.</td>
</tr>
<tr>
<td><code>-z, --authorization &lt;type&gt;</code></td>
<td>Specifies authorization type. type can be 'self', 'host', 'none', or a string specifying the expected identity of the remote party.</td>
</tr>
<tr>
<td><code>-t, --timeout &lt;timeout&gt;</code></td>
<td>Specifies client timeout (in seconds). The client will wait maximum of the timeout value for a response from the server before returning an error. By default the timeout value is 10 minutes.</td>
</tr>
</tbody>
</table>

Example:

```
$ wsn-subscribe -s http://localhost:8080/wsrfservices/CounterService \ -k "({http://counter.com}CounterKey" 123 \ "({http://counter.com}Value"
```
**Name**

globus-deploy-gar -- Deploys a GAR file (locally)

globus-deploy-gar

**Tool description**

Deploys a GAR file (locally) into Java WS Core or Apache Tomcat container.

**Command syntax**

globus-deploy-gar [options] <gar.file>

The `<gar.file>` is the path to the GAR file to be deployed.

**Table 22. Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-help</code></td>
<td>Displays help information about the command.</td>
</tr>
<tr>
<td><code>-debug</code></td>
<td>Enables debug mode.</td>
</tr>
<tr>
<td><code>-verbose</code></td>
<td>Enables verbose mode.</td>
</tr>
<tr>
<td><code>-backup</code></td>
<td>Creates backup of existing configuration files.</td>
</tr>
<tr>
<td><code>-overwrite</code></td>
<td>Overwrite existing deployment.</td>
</tr>
<tr>
<td><code>-profile</code></td>
<td>Specifies the profile name under which the configuration files in the GAR will be deployed. Please see &quot;Configuration Profiles&quot; under Configuring Java WS Core for details.</td>
</tr>
<tr>
<td><code>-tomcat</code></td>
<td>Deploys a GAR file to Apache Tomcat. The <code>&lt;dir&gt;</code> argument must point to the Tomcat installation directory. Note: Java WS Core must be already deployed in Tomcat. Please see Deploying into Tomcat section for details.</td>
</tr>
<tr>
<td><code>-D&lt;property&gt;=&lt;value&gt;</code></td>
<td>Passes arbitrary property-value pairs. See below for the list of currently supported properties.</td>
</tr>
</tbody>
</table>

**Table 23. Supported property-value pairs**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-Dall.scripts=true</code></td>
<td>Causes Windows and Unix launcher scripts to be generated.</td>
</tr>
<tr>
<td><code>-DDdoValidation=false</code></td>
<td>Turns off automatic validation of service configuration files.</td>
</tr>
</tbody>
</table>

**Note**

Since GT 4.2, `globus-deploy-gar` command will NOT overwrite the existing deployment unless `-overwrite` option is specified. It is recommended to undeploy the existing deployment first. The container must be off to deploy a GAR file.

Example I:

```
$ globus-deploy-gar /tmp/gars/globus_wsrf_core_samples_counter.gar
```
The above command will deploy the `globus_wsrf_core_samples_counter.gar` into Java WS Core installation directory. The above command invokes the deployGar task in the `build-packages.xml` Ant build file. The above example is equivalent to running:

```
$ ant -f $GLOBUS_LOCATION/share/globus_wsrf_common/build-packages.xml deployGar \
   -Dgar.name=/tmp/gars/globus_wsrf_core_samples_counter.gar
```

The profile name can be passed using the `-Dprofile` Ant option. To enable back up of the existing configuration files add the `-DcreateBackup=true` Ant option. Make sure to use the `absolute` path name for the gar file when using Ant directly.

Example II:

```
$ globus-deploy-gar -tomcat /soft/tomcat-5.5.20 \
   /tmp/gars/globus_wsrf_core_samples_counter.gar
```

The above command will deploy the `globus_wsrf_core_samples_counter.gar` into Apache Tomcat. The above command invokes the deployGar task in the `tomcat-service.xml` Ant build file. The above example is equivalent to running:

```
$ ant -f $GLOBUS_LOCATION/share/globus_wsrf_common/tomcat/tomcat-service.xml deployGar \
   -Dgar.name=/tmp/gars/globus_wsrf_core_samples_counter.gar \
   -Dtomcat.dir=/soft/tomcat-5.5.20
```

By default the GAR file will be deployed under the "wsrf" web application. To specify a different web application name use `-Dwebapp.name=<name>` option.
Name

globus-undeploy-gar -- Undeploys a GAR file (locally)

globus-undeploy-gar

Tool description

Undeploys a GAR file (locally) from Java WS Core or Apache Tomcat container.

Command syntax

globus-undeploy-gar [options] <gar.id>

The <gar.id> is the base name of the GAR file without the .gar extension to undeploy. For example if the GAR file is "foo.gar", then the GAR id is "foo". The directory names under $GLOBUS_LOCATION/etc/globus_packages/ are the GAR ids of the undeployable items.

Table 24. Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-help</td>
<td>Displays help information about the command.</td>
</tr>
<tr>
<td>-debug</td>
<td>Enables debug mode.</td>
</tr>
<tr>
<td>-verbose</td>
<td>Enables verbose mode.</td>
</tr>
<tr>
<td>-tomcat &lt;dir&gt;</td>
<td>Undeploy a GAR file from Apache Tomcat. The &lt;dir&gt; argument must point to the Tomcat installation directory.</td>
</tr>
<tr>
<td>-D&lt;property&gt;=&lt;value&gt;</td>
<td>Passes arbitrary property-value pairs.</td>
</tr>
</tbody>
</table>

Note

The container must be off to undeploy a GAR file.

Example I:

```
$ globus-undeploy-gar globus_wsrf_core_samples_counter
```

The above command will undeploy globus_wsrf_core_samples_counter GAR from Java WS Core installation directory. The above command invokes the undeployGar task in the build-packages.xml Ant build file. The above example is equivalent to running:

```
$ ant -f $GLOBUS_LOCATION/share/globus_wsrf_common/build-packages.xml undeployGar \
   -Dgar.id=globus_wsrf_core_samples_counter
```

Example II:

```
$ globus-undeploy-gar -tomcat /soft/tomcat-5.5.20 \ globus_wsrf_core_samples_counter
```

The above command will undeploy globus_wsrf_core_samples_counter GAR from Apache Tomcat. The above command invokes the undeployGar task in the tomcat-service.xml Ant build file. The above example is equivalent to running:

```
$ ant -f $GLOBUS_LOCATION/share/globus_wsrf_common/tomcat/tomcat-service.xml undeployGar \
   -Dgar.id=globus_wsrf_core_samples_counter \ -Dtomcat.dir=/soft/tomcat-5.5.20
```
By default the GAR file will be undeployed under the "wsrf" web application. To specify a different web application name use `-Dwebapp.name=<name>` option.
Name

globus-check-environment -- Displays component version information and validates JVM version.

globus-check-environment

Tool description

Displays component version information and validates the JVM version. This tool is primarily used for debugging purposes.
**Name**

globus-check-remote-environment -- Displays remote component version information.

globus-check-remote-environment

**Tool description**

Displays remote component version information.

**Command syntax**

`globus-check-remote-environment [-help] -s endpoint -z authz`

**Table 25. Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-help</code></td>
<td>Displays help information about the command.</td>
</tr>
<tr>
<td><code>-s endpoint</code></td>
<td>Remote endpoint to print version information about. It should be of the format protocol://host:port, example <a href="https://localhost:8443">https://localhost:8443</a>.</td>
</tr>
<tr>
<td><code>-z authz</code></td>
<td>Sets authorization, can be 'self', 'host', 'hostOrSelf' or 'none' or a string specifying the expected identity of the remote party. Defaults to no authorization.</td>
</tr>
</tbody>
</table>
Name

globus-update-client-config -- Merges client-config.wsdd files from deployed modules into the global client-config.wsdd configuration file

globus-update-client-config

Tool description

Merges multiple client-config.wsdd files from deployed modules into the global configuration file. Scans each $GLOBUS_LOCATION/etc/<modulename>/client-config.wsdd and merges the contents into $GLOBUS_LOCATION/client-config.wsdd. This tool is primarily intended for use by administrators and automation tools to facilitate the adding and removing of module specific type-mapping and/or other client-side configuration from the global client-config.wsdd file used by the Globus installation.

Command syntax

globus-update-client-config [<filename>]

Table 26. Options

<table>
<thead>
<tr>
<th>&lt;filename&gt;</th>
<th>Optional argument that specifies an alternate path to write the result client-config.wsdd file. By default, running the program with no arguments will write the file to $GLOBUS_LOCATION/client-config.wsdd</th>
</tr>
</thead>
</table>

85
Name

globus-validate-descriptors -- Validate configuration files of all services

globus-validate-descriptors

Tool description

Validates the Web Services Deployment Descriptor (.wsdd) files, JNDI configuration files (jndi-config.xml), and security descriptors for all services.

Command syntax

globus-validate-descriptors [options]

Table 27. Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-help</td>
<td>Displays help information about the command.</td>
</tr>
<tr>
<td>-debug</td>
<td>Enables debug mode.</td>
</tr>
<tr>
<td>-verbose</td>
<td>Enables verbose mode.</td>
</tr>
<tr>
<td>-D&lt;property&gt;=&lt;value&gt;</td>
<td>Passes arbitrary property-value pairs.</td>
</tr>
</tbody>
</table>
Name
globus-reload-container -- Reinitializes standalone container
globus-reload-container

Tool description
Invokes the reload() operation on the DeployService running in the remote container. It tells the container to reinitialize all of its services, re-read its and service configuration files, etc. For example, the administrator can change the security descriptor of a service and then use the globus-reload-container command to force the container to load the updated configuration without restarting the container.

By default the DeployService is configured to perform self authorization and therefore the globus-reload-container must be executed with the same credentials as the container it is running with. Alternatively, the service can be configured with a gridmap file to allow a subset of users (with their own credentials) to invoke the service (please see the service security deployment descriptor for details).

Note
This command only works with the standalone container. Please see the Java WS Core Dynamic Deploy Design Document for more information.

Command syntax
globus-reload-container [options]
Table 28. Common options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Displays help information about the command.</td>
</tr>
<tr>
<td>-d, --debug</td>
<td>Enables debug mode. For example, full stack traces of errors will be displayed.</td>
</tr>
<tr>
<td>-e, --eprFile &lt;file&gt;</td>
<td>Specifies an XML file that contains the WS-Addressing endpoint reference.</td>
</tr>
<tr>
<td>-s, --service &lt;url&gt;</td>
<td>Specifies the service URL.</td>
</tr>
<tr>
<td>-k, --key &lt;name value&gt;</td>
<td>Specifies the resource key. The name is the QName of the resource key in the string form: {namespaceURI}localPart, while the value is the simple value of the key. For complex keys, use the --eprFile option. Example: <code>-k &quot;http://www.globus.org)MyKey&quot;</code></td>
</tr>
<tr>
<td>-f, --descriptor &lt;file&gt;</td>
<td>Specifies a client security descriptor. Overrides all other security settings.</td>
</tr>
<tr>
<td>-a, --anonymous</td>
<td>Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-g, --delegation &lt;mode&gt;</td>
<td>Enables delegation. mode can be either 'limited' or 'full'. Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-l, --contextLifetime &lt;value&gt;</td>
<td>Sets the lifetime of the client security context. value is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-m, --securityMech &lt;type&gt;</td>
<td>Specifies the authentication mechanism. type can be 'msg' for GSI Secure Message, or 'conv' for GSI Secure Conversation.</td>
</tr>
<tr>
<td>-c, --serverCertificate &lt;file&gt;</td>
<td>Specifies the server's certificate file used for encryption. Only needed for the GSI Secure Message authentication mechanism.</td>
</tr>
<tr>
<td>-p, --protection &lt;type&gt;</td>
<td>Specifies the protection level. type can be 'sig' for signature or 'enc' for encryption.</td>
</tr>
<tr>
<td>-x, --proxyFilename &lt;value&gt;</td>
<td>Sets the proxy file to use as client credential.</td>
</tr>
<tr>
<td>-z, --authorization &lt;type&gt;</td>
<td>Specifies authorization type. type can be 'self', 'host', 'none', or a string specifying the expected identity of the remote party.</td>
</tr>
<tr>
<td>-t, --timeout &lt;timeout&gt;</td>
<td>Specifies client timeout (in seconds). The client will wait maximum of the timeout value for a response from the server before returning an error. By default the timeout value is 10 minutes.</td>
</tr>
</tbody>
</table>

Example:

```
$ globus-reload-container
```
Name

globus-remote-undeploy-gar -- Undeploys a GAR file (remotely)

globus-remote-undeploy-gar

Tool description

The `globus-remote-undeploy-gar` command undeploys a GAR file remotely. It invokes the `undeploy()` operation on the `DeployService` running in the remote container. It works just like the `globus-undeploy-gar` command but the GAR file is undeployed remotely.

By default the `DeployService` is configured to perform `self` authorization and therefore the `globus-remote-undeploy-gar` must be executed with the same credentials as the container it is running with. Alternatively, the service can be configured with a gridmap file to allow a subset of users (with their own credentials) to invoke the service (please see the service security deployment descriptor for details).

Note

This command only works with the standalone container. Please see the Java WS Core Dynamic Deploy Design Document for more information.

Command syntax

globus-remote-undeploy-gar [options] <gar.id>

The `<gar.id>` is the base name of the GAR file without the `.gar` extension to undeploy. For example if the GAR file is "foo.gar", then the GAR id is "foo".
Table 29. Common options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Displays help information about the command.</td>
</tr>
<tr>
<td>-d, --debug</td>
<td>Enables debug mode. For example, full stack traces of errors will be dis-</td>
</tr>
<tr>
<td></td>
<td>played.</td>
</tr>
<tr>
<td>-e, --eprFile &lt;file&gt;</td>
<td>Specifies an XML file that contains the WS-Addressing endpoint reference.</td>
</tr>
<tr>
<td>-s, --service &lt;url&gt;</td>
<td>Specifies the service URL.</td>
</tr>
<tr>
<td>-k, --key &lt;name value&gt;</td>
<td>Specifies the resource key. The name is the QName of the resource key in</td>
</tr>
<tr>
<td></td>
<td>the string form: {namespaceURI}localPart, while the value is the simple</td>
</tr>
<tr>
<td></td>
<td>value of the key. For complex keys, use the --eprFile option. Example:</td>
</tr>
<tr>
<td></td>
<td>-k &quot;(<a href="http://www.globus.org)MyKey">http://www.globus.org)MyKey</a>&quot;</td>
</tr>
<tr>
<td>-f, --descriptor &lt;file&gt;</td>
<td>Specifies a client security descriptor. Overrides all other security</td>
</tr>
<tr>
<td></td>
<td>settings.</td>
</tr>
<tr>
<td>-a, --anonymous</td>
<td>Enables anonymous authentication. Only supported with transport security</td>
</tr>
<tr>
<td></td>
<td>or the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-g, --delegation &lt;mode&gt;</td>
<td>Enables delegation. mode can be either 'limited' or 'full'. Only supported</td>
</tr>
<tr>
<td></td>
<td>with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-l, --contextLifetime &lt;value&gt;</td>
<td>Sets the lifetime of the client security context. value is in milliseconds.</td>
</tr>
<tr>
<td></td>
<td>Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-m, --securityMech &lt;type&gt;</td>
<td>Specifies the authentication mechanism. type can be 'msg' for(GSI Secure</td>
</tr>
<tr>
<td></td>
<td>Message, or 'conv' for GSI Secure Conversation.</td>
</tr>
<tr>
<td>-c, --serverCertificate &lt;file&gt;</td>
<td>Specifies the server's certificate file used for encryption. Only needed for</td>
</tr>
<tr>
<td></td>
<td>the GSI Secure Message authentication mechanism.</td>
</tr>
<tr>
<td>-p, --protection &lt;type&gt;</td>
<td>Specifies the protection level. type can be 'sig' for signature or 'enc'</td>
</tr>
<tr>
<td></td>
<td>for encryption.</td>
</tr>
<tr>
<td>-x, --proxyFilename &lt;value&gt;</td>
<td>Sets the proxy file to use as client credential.</td>
</tr>
<tr>
<td>-z, --authorization &lt;type&gt;</td>
<td>Specifies authorization type. type can be 'self', 'host', 'none', or a</td>
</tr>
<tr>
<td></td>
<td>string specifying the expected identity of the remote party.</td>
</tr>
<tr>
<td>-t, --timeout &lt;timeout&gt;</td>
<td>Specifies client timeout (in seconds). The client will wait maximum of</td>
</tr>
<tr>
<td></td>
<td>the timeout value for a response from the server before returning an error.</td>
</tr>
<tr>
<td></td>
<td>By default the timeout value is 10 minutes.</td>
</tr>
</tbody>
</table>

Example:

$ globus-remote-undeploy-gar globus_wsrfsr_core_samples_counter

To see what GAR files can be undeployed on the remote container run the following query on the DeployService, for example:

$ wsrfsr-query -z hostSelf -s https://127.0.0.1:8443/wsrf/services/DeployService
Name

globus-remote-deploy-gar -- Deploys a GAR file (remotely)

globus-remote-deploy-gar

Tool description

The globus-remote-deploy-gar command deploys a GAR file remotely. It first transfers the GAR file to the DeployService running in the remote container and then it deploys it using the deploy() operation of the service (the tool can also perform these two operations separately).

By default the DeployService is configured to perform self authorization and therefore the globus-remote-deploy-gar must be executed with the same credentials as the container it is running with. Alternatively, the service can be configured with a gridmap file to allow a subset of users (with their own credentials) to invoke the service (please see the service security deployment descriptor for details).

Note

This command only works with the standalone container. Please see the Java WS Core Dynamic Deploy Design Document for more information.

Command syntax

globus-remote-deploy-gar [options] <gar>

The <gar> can be either an URL or a file location. If a file location is passed to the tool, it will transfer the file to the service via SOAP with Attachments (the upload() function) using the MTOM format. If an URL is passed, the tool will call the download() function of the service, and let the service download the GAR file.

The following are command-specific options in addition to the common options:

Table 30. Command-specific options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-n, --transfer</td>
<td>Transfer GAR file only.</td>
</tr>
<tr>
<td>-y, --deploy</td>
<td>Deploy GAR file only (assumes the GAR is already transferred to the DeployService.</td>
</tr>
<tr>
<td>-o, --overwrite</td>
<td>Overwrite existing deployment.</td>
</tr>
<tr>
<td>-b, --backup</td>
<td>Creates backup of existing configuration files</td>
</tr>
</tbody>
</table>
Table 31. Common options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Displays help information about the command.</td>
</tr>
<tr>
<td>-d, --debug</td>
<td>Enables debug mode. For example, full stack traces of errors will be dis-</td>
</tr>
<tr>
<td></td>
<td>played.</td>
</tr>
<tr>
<td>-e, --eprFile &lt;file&gt;</td>
<td>Specifies an XML file that contains the WS-Addressing endpoint reference.</td>
</tr>
<tr>
<td>-s, --service &lt;url&gt;</td>
<td>Specifies the service URL.</td>
</tr>
<tr>
<td>-k, --key &lt;name value&gt;</td>
<td>Specifies the resource key. The name is the QName of the resource key</td>
</tr>
<tr>
<td></td>
<td>in the string form: namespaceURI</td>
</tr>
<tr>
<td></td>
<td>-k &quot;<a href="http://www.globus.org">http://www.globus.org</a></td>
</tr>
<tr>
<td>-f, --descriptor &lt;file&gt;</td>
<td>Specifies a client security descriptor. Overrides all other security settings.</td>
</tr>
<tr>
<td>-a, --anonymous</td>
<td>Enables anonymous authentication. Only supported with transport security or</td>
</tr>
<tr>
<td></td>
<td>the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-g, --delegation &lt;mode&gt;</td>
<td>Enables delegation. mode can be either 'limited' or 'full'. Only supported</td>
</tr>
<tr>
<td></td>
<td>with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-l, --contextLifetime &lt;value&gt;</td>
<td>Sets the lifetime of the client security context. value is in milliseconds.</td>
</tr>
<tr>
<td></td>
<td>Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-m, --securityMech &lt;type&gt;</td>
<td>Specifies the authentication mechanism. type can be 'msg' for GSI Secure</td>
</tr>
<tr>
<td></td>
<td>Message, or 'conv' for GSI Secure Conversation.</td>
</tr>
<tr>
<td>-c, --serverCertificate &lt;file&gt;</td>
<td>Specifies the server's certificate file used for encryption. Only needed for</td>
</tr>
<tr>
<td></td>
<td>the GSI Secure Message authentication mechanism.</td>
</tr>
<tr>
<td>-p, --protection &lt;type&gt;</td>
<td>Specifies the protection level. type can be 'sig' for signature or 'enc' for</td>
</tr>
<tr>
<td></td>
<td>encryption.</td>
</tr>
<tr>
<td>-x, --proxyFilename &lt;value&gt;</td>
<td>Sets the proxy file to use as client credential.</td>
</tr>
<tr>
<td>-z, --authorization &lt;type&gt;</td>
<td>Specifies authorization type. type can be 'self', 'host', 'none', or a string</td>
</tr>
<tr>
<td></td>
<td>specifying the expected identity of the remote party.</td>
</tr>
<tr>
<td>-t, --timeout &lt;timeout&gt;</td>
<td>Specifies client timeout (in seconds). The client will wait maximum of the</td>
</tr>
<tr>
<td></td>
<td>timeout value for a response from the server before returning an error. By</td>
</tr>
<tr>
<td></td>
<td>default the timeout value is 10 minutes.</td>
</tr>
</tbody>
</table>

Examples:

$ globus-remote-deploy-gar /tmp/myServie.gar

$ globus-remote-deploy-gar gsiftp://localhost/tmp/myServie.gar

To see what GAR files haven been transfered but not yet deployed on the remote container run the following query on the DeployService, for example:

$ wsrf-query -z hostSelf -s https://127.0.0.1:8443/wsrf/services/DeployService
Name

ws-enumerate-start -- Starts an enumeration

ws-enumerate-start

Tool description

Creates a new enumeration context and prints it out to the console.

Note

The remote service must support the enumerate operation of the WS-Enumeration specification.

Command syntax

ws-enumerate-start [options]
Table 32. Common options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Displays help information about the command.</td>
</tr>
<tr>
<td>-d, --debug</td>
<td>Enables debug mode. For example, full stack traces of errors will be displayed.</td>
</tr>
<tr>
<td>-e, --eprFile &lt;file&gt;</td>
<td>Specifies an XML file that contains the <strong>WS-Addressing</strong> endpoint reference.</td>
</tr>
<tr>
<td>-s, --service &lt;url&gt;</td>
<td>Specifies the service URL.</td>
</tr>
<tr>
<td>-k, --key &lt;name value&gt;</td>
<td>Specifies the resource key. The <strong>name</strong> is the QName of the resource key in the string form: <code>{namespaceURI}localPart</code>, while the <strong>value</strong> is the simple value of the key. For complex keys, use the <strong>--eprFile</strong> option. Example:</td>
</tr>
<tr>
<td></td>
<td><code>-k &quot;{http://www.globus.org}MyKey&quot; 123</code></td>
</tr>
<tr>
<td>-f, --descriptor &lt;file&gt;</td>
<td>Specifies a client security descriptor. Overrides all other security settings.</td>
</tr>
<tr>
<td>-a, --anonymous</td>
<td>Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-g, --delegation &lt;mode&gt;</td>
<td>Enables delegation. <strong>mode</strong> can be either 'limited' or 'full'. Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-l, --contextLifetime &lt;value&gt;</td>
<td>Sets the lifetime of the client security context. <strong>value</strong> is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-m, --securityMech &lt;type&gt;</td>
<td>Specifies the authentication mechanism. <strong>type</strong> can be 'msg' for GSI Secure Message, or 'conv' for GSI Secure Conversation.</td>
</tr>
<tr>
<td>-c, --serverCertificate &lt;file&gt;</td>
<td>Specifies the server's <strong>certificate</strong> file used for encryption. Only needed for the GSI Secure Message authentication mechanism.</td>
</tr>
<tr>
<td>-p, --protection &lt;type&gt;</td>
<td>Specifies the protection level. <strong>type</strong> can be 'sig' for signature or 'enc' for encryption.</td>
</tr>
<tr>
<td>-x, --proxyFilename &lt;value&gt;</td>
<td>Sets the proxy file to use as client credential.</td>
</tr>
<tr>
<td>-z, --authorization &lt;type&gt;</td>
<td>Specifies authorization type. <strong>type</strong> can be 'self', 'host', 'none', or a string specifying the expected identity of the remote party.</td>
</tr>
<tr>
<td>-t, --timeout &lt;timeout&gt;</td>
<td>Specifies client timeout (in seconds). The client will wait maximum of the timeout value for a response from the server before returning an error. By default the timeout value is 10 minutes.</td>
</tr>
</tbody>
</table>

Example:

```bash
$ ws-enumerate-start -s http://localhost:8080/wsrfservices/ContainerRegistryService > enum.context
```

The created enumeration context will be stored in the `enum.context` file which then can be passed to `ws-enumerate` and `ws-enumerate-end` command line clients.
Name

ws-enumerate -- Retrieves enumeration data

ws-enumerate

Tool description

Retrieves the next set of enumeration data and prints it out to the console.

Note

The remote service must implement the WS-Enumeration specification.

Command syntax

ws-enumerate [options] <enumContextFile>

The <enumContextFile> is a file that contains the enumeration context.

The following are command-specific options in addition to the common options:

Table 33. Command-specific options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-i, --items &lt;int&gt;</td>
<td>Specifies the total number of enumeration items to retrieve. The parameter value can be 'all' to retrieve all enumeration data. By default, only one element is retrieved.</td>
</tr>
<tr>
<td>-r, --maxCharacters &lt;int&gt;</td>
<td>Specifies the maximum number of characters (in Unicode) of the enumeration data that the client can accept at a time. By default, there is no limit on the size of the elements.</td>
</tr>
<tr>
<td>-n, --maxElements &lt;int&gt;</td>
<td>Specifies the maximum number of enumeration items to fetch at a time. By default, one element is retrieved at a time.</td>
</tr>
<tr>
<td>-o, --maxTime &lt;int&gt;</td>
<td>Specifies the maximum amount of time (in milliseconds) in which the enumeration data must be assembled. By default, there is no time limit.</td>
</tr>
</tbody>
</table>
Table 34. Common options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>-h, --help</strong></td>
<td>Displays help information about the command.</td>
</tr>
<tr>
<td><strong>-d, --debug</strong></td>
<td>Enables debug mode. For example, full stack traces of errors will be displayed.</td>
</tr>
<tr>
<td><strong>-e, --eprFile &lt;file&gt;</strong></td>
<td>Specifies an XML file that contains the WS-Addressing endpoint reference.</td>
</tr>
<tr>
<td><strong>-s, --service &lt;url&gt;</strong></td>
<td>Specifies the service URL.</td>
</tr>
</tbody>
</table>
| **-k, --key <name value>** | Specifies the resource key. The name is the QName of the resource key in the string form: `{namespaceURI}localPart`, while the value is the simple value of the key. For complex keys, use the --eprFile option. Example: 
```
-k "(http://www.globus.org)MyKey"
```
| **-f, --descriptor <file>** | Specifies a client security descriptor. Overrides all other security settings. |
| **-a, --anonymous** | Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism. |
| **-g, --delegation <mode>** | Enables delegation. mode can be either 'limited' or 'full'. Only supported with the GSI Secure Conversation authentication mechanism. |
| **-l, --contextLifetime <value>** | Sets the lifetime of the client security context. value is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism. |
| **-m, --securityMech <type>** | Specifies the authentication mechanism. type can be 'msg' for GSI Secure Message, or 'conv' for GSI Secure Conversation. |
| **-c, --serverCertificate <file>** | Specifies the server’s certificate file used for encryption. Only needed for the GSI Secure Message authentication mechanism. |
| **-p, --protection <type>** | Specifies the protection level. type can be 'sig' for signature or 'enc' for encryption. |
| **-x, --proxyFilename <value>** | Sets the proxy file to use as client credential. |
| **-z, --authorization <type>** | Specifies authorization type. type can be 'self', 'host', 'none', or a string specifying the expected identity of the remote party. |
| **-t, --timeout <timeout>** | Specifies client timeout (in seconds). The client will wait maximum of the timeout value for a response from the server before returning an error. By default the timeout value is 10 minutes. |

Example:
```
$ ws-enumerate -s http://localhost:8080/wsrfservices/ContainerRegistryService \ -i 10 \ -n 5 enum.context
```

This command will display 10 elements of the enumeration data obtaining 5 elements at a time from the service.
Name

ws-enumerate-end -- Stops an enumeration

ws-enumerate-end

Tool description

Releases an enumeration context.

Note

The remote service must implement the WS-Enumeration specification.

Command syntax

ws-enumerate-end [options] <enumContextFile>

The <enumContextFile> is a file that contains the enumeration context.
## Table 35. Common options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-h, --help</code></td>
<td>Displays help information about the command.</td>
</tr>
<tr>
<td><code>-d, --debug</code></td>
<td>Enables debug mode. For example, full stack traces of errors will be displayed.</td>
</tr>
<tr>
<td><code>-e, --eprFile &lt;file&gt;</code></td>
<td>Specifies an XML file that contains the WS-Addressing endpoint reference.</td>
</tr>
<tr>
<td><code>-s, --service &lt;url&gt;</code></td>
<td>Specifies the service URL.</td>
</tr>
<tr>
<td><code>-k, --key &lt;name value&gt;</code></td>
<td>Specifies the resource key. The name is the QName of the resource key in the string form: <code>{namespaceURI}localPart</code>, while the value is the simple value of the key. For complex keys, use the <code>--eprFile</code> option. Example: <code>-k &quot;\{http://www.globus.org\}MyKey&quot; 123</code></td>
</tr>
<tr>
<td><code>-f, --descriptor &lt;file&gt;</code></td>
<td>Specifies a client security descriptor. Overrides all other security settings.</td>
</tr>
<tr>
<td><code>-a, --anonymous</code></td>
<td>Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td><code>-g, --delegation &lt;mode&gt;</code></td>
<td>Enables delegation. mode can be either 'limited' or 'full'. Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td><code>-l, --contextLifetime &lt;value&gt;</code></td>
<td>Sets the lifetime of the client security context. value is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td><code>-m, --securityMech &lt;type&gt;</code></td>
<td>Specifies the authentication mechanism. type can be 'msg' for GSI Secure Message, or 'conv' for GSI Secure Conversation.</td>
</tr>
<tr>
<td><code>-c, --serverCertificate &lt;file&gt;</code></td>
<td>Specifies the server's certificate file used for encryption. Only needed for the GSI Secure Message authentication mechanism.</td>
</tr>
<tr>
<td><code>-p, --protection &lt;type&gt;</code></td>
<td>Specifies the protection level. type can be 'sig' for signature or 'enc' for encryption.</td>
</tr>
<tr>
<td><code>-x, --proxyFilename &lt;value&gt;</code></td>
<td>Sets the proxy file to use as client credential.</td>
</tr>
<tr>
<td><code>-z, --authorization &lt;type&gt;</code></td>
<td>Specifies authorization type. type can be 'self', 'host', 'none', or a string specifying the expected identity of the remote party.</td>
</tr>
<tr>
<td><code>-t, --timeout &lt;timeout&gt;</code></td>
<td>Specifies client timeout (in seconds). The client will wait maximum of the timeout value for a response from the server before returning an error. By default the timeout value is 10 minutes.</td>
</tr>
</tbody>
</table>

Example:

```
\$ ws-enumerate-end -s http://localhost:8080/wsrf/services/ContainerRegistryService enum.context
```
Name

globus-xpath-query -- Performs XPath query on a resource property document

globus-xpath-query

Tool description

The globus-xpath-query uses a custom query dialect implementation called TargetedXPath to query the resource property document of a resource. Please see the querying resource properties using XPath section for more details.

Command syntax

globus-xpath-query [options] [query expression] [rpQName]

The query expression is an XPath expression. The rpQName is a resource property QName. If a resource property is specified only that resource property within the resource property document will be queried. Otherwise, the entire resource property document will be queried. By default, a simple XPath query is assumed that returns the entire resource property document.

Table 36. Command-specific options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-n, --nsMapFile &lt;file&gt;</td>
<td>Specifies a file that contains namespace mappings. By default, the etc/globus_wsrf_core/namespace-mappings.xml file is used.</td>
</tr>
<tr>
<td>-u, --enumerate</td>
<td>Enumerate the query results. The query response will contain an enumeration context through which the actual query results can be obtained. The returned enumeration context can be used with the ws-enumerate command line tool. Also, please note that by default the enumeration context will expire in 30 minutes.</td>
</tr>
</tbody>
</table>
Table 37. Common options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Displays help information about the command.</td>
</tr>
<tr>
<td>-d, --debug</td>
<td>Enables debug mode. For example, full stack traces of errors will be displayed.</td>
</tr>
<tr>
<td>-e, --eprFile &lt;file&gt;</td>
<td>Specifies an XML file that contains the WS-Addressing endpoint reference.</td>
</tr>
<tr>
<td>-s, --service &lt;url&gt;</td>
<td>Specifies the service URL.</td>
</tr>
</tbody>
</table>
| -k, --key <name value> | Specifies the resource key. The name is the QName of the resource key in the string form: namespaceURIlocalPart, while the value is the simple value of the key. For complex keys, use the --eprFile option. Example: 
- k "(http://www.globus.org)MyKey"
| -f, --descriptor <file> | Specifies a client security descriptor. Overrides all other security settings.                                                            |
| -a, --anonymous | Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism.          |
| -g, --delegation <mode> | Enables delegation. mode can be either 'limited' or 'full'. Only supported with the GSI Secure Conversation authentication mechanism.  |
| -l, --contextLifetime <value> | Sets the lifetime of the client security context. value is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism. |
| -m, --securityMech <type> | Specifies the authentication mechanism. type can be 'msg' for GSI Secure Message, or 'conv' for GSI Secure Conversation.               |
| -c, --serverCertificate <file> | Specifies the server's certificate file used for encryption. Only needed for the GSI Secure Message authentication mechanism.             |
| -p, --protection <type> | Specifies the protection level. type can be 'sig' for signature or 'enc' for encryption.                                                  |
| -x, --proxyFilename <value> | Sets the proxy file to use as client credential.                                                                                          |
| -z, --authorization <type> | Specifies authorization type. type can be 'self', 'host', 'none', or a string specifying the expected identity of the remote party.     |
| -t, --timeout <timeout> | Specifies client timeout (in seconds). The client will wait maximum of the timeout value for a response from the server before returning an error. By default the timeout value is 10 minutes. |

Examples:

```
$ globus-xpath-query -s http://localhost:8080/wsrfservices/ContainerRegistryService \
    "//wssg:MemberServiceEPR/wsa:Address"
```

The above command will query the entire resource property document of the service.

```
$ globus-xpath-query -s http://localhost:8080/wsrfservices/ContainerRegistryService \
    "//wssg:MemberServiceEPR/wsa:Address" wssg:Entry
```

The above command will query only the wssg:Entry resource property of the resource property document of the service.

The first command will create an enumeration for the query results and store the returned enumeration context in a file. The second command will use the enumeration context stored in that file to retrieve the actual query results.
Name
Common Java Client Options -- list of common options across commands

Common Java Client Options

Table 38. Common options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Displays help information about the command.</td>
</tr>
<tr>
<td>-d, --debug</td>
<td>Enables debug mode. For example, full stack traces of errors will be displayed.</td>
</tr>
<tr>
<td>-e, --eprFile &lt;file&gt;</td>
<td>Specifies an XML file that contains the WS-Addressing endpoint reference.</td>
</tr>
<tr>
<td>-s, --service &lt;url&gt;</td>
<td>Specifies the service URL.</td>
</tr>
<tr>
<td>-k, --key &lt;name value&gt;</td>
<td>Specifies the resource key. The name is the QName of the resource key in the string form: [namespaceURI]localPart, while the value is the simple value of the key. For complex keys, use the --eprFile option. Example: -k &quot;{&quot;<a href="http://www.globus.org%7DMyKey">http://www.globus.org}MyKey</a>&quot; 123</td>
</tr>
<tr>
<td>-f, --descriptor &lt;file&gt;</td>
<td>Specifies a client security descriptor. Overrides all other security settings.</td>
</tr>
<tr>
<td>-a, --anonymous</td>
<td>Enables anonymous authentication. Only supported with transport security or the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-g, --delegation &lt;mode&gt;</td>
<td>Enables delegation. mode can be either 'limited' or 'full'. Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-l, --contextLifetime &lt;value&gt;</td>
<td>Sets the lifetime of the client security context. value is in milliseconds. Only supported with the GSI Secure Conversation authentication mechanism.</td>
</tr>
<tr>
<td>-m, --securityMech &lt;type&gt;</td>
<td>Specifies the authentication mechanism. type can be 'msg' for GSI Secure Message, or 'conv' for GSI Secure Conversation.</td>
</tr>
<tr>
<td>-c, --serverCertificate &lt;file&gt;</td>
<td>Specifies the server's certificate file used for encryption. Only needed for the GSI Secure Message authentication mechanism.</td>
</tr>
<tr>
<td>-p, --protection &lt;type&gt;</td>
<td>Specifies the protection level. type can be 'sig' for signature or 'enc' for encryption.</td>
</tr>
<tr>
<td>-x, --proxyFilename &lt;value&gt;</td>
<td>Sets the proxy file to use as client credential.</td>
</tr>
<tr>
<td>-z, --authorization &lt;type&gt;</td>
<td>Specifies authorization type. type can be 'self', 'host', 'none', or a string specifying the expected identity of the remote party.</td>
</tr>
<tr>
<td>-t, --timeout &lt;timeout&gt;</td>
<td>Specifies client timeout (in seconds). The client will wait maximum of the timeout value for a response from the server before returning an error. By default the timeout value is 10 minutes.</td>
</tr>
</tbody>
</table>
Chapter 8. Configuring

1. Configuration overview

Java WS Core provides per-service configuration and supports configuration profiles. The configuration information of a service is mainly encapsulated in two separate configuration files:

- `server-config.wsdd` (Web Service Deployment Descriptor) - contains information about the web service.
- `jndi-config.xml` (JNDI configuration file) - contains information about the resource management.

A service that supports security might also have the `security-config.xml` (security deployment descriptor) file. Please see Java WS A&A Security Descriptor Framework for details.

All these configuration files are dropped into the `$GLOBUS_LOCATION/etc/<gar.id>/` directory during the deployment process.

2. Syntax of the interface:

2.1. Global Configuration

The global properties are specified in the `<globalConfiguration>` section of `*server-config.wsdd` files in the `$GLOBUS_LOCATION/etc/globus_wsrf_core/` directory. The configuration item `name` corresponds to the "name" attribute in a `<parameter>` sub element, and the `value` is put as a "value" attribute within the same parameter element.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>logicalHost</td>
<td>&lt;hostname&gt;</td>
<td>This parameter specifies the hostname to use instead of the default local host. It is equivalent to setting the GLOBUS_HOSTNAME environment property. Can be FQDN or just hostname.</td>
<td>Optional</td>
</tr>
<tr>
<td>disableDNS</td>
<td>&lt;boolean&gt;</td>
<td>This parameter specifies whether to perform DNS lookup on the logicalHost parameter. By default, &quot;false&quot; is assumed (DNS lookup is performed).</td>
<td>Optional</td>
</tr>
<tr>
<td>domainName</td>
<td>&lt;domain name&gt;</td>
<td>This parameter specifies the domain name to append to the host name if the host name is not qualified by a domain.</td>
<td>Optional</td>
</tr>
<tr>
<td>publishHostName</td>
<td>&lt;boolean&gt;</td>
<td>This parameter specifies whether to publish the hostname or the ip address. It is only used when DNS lookups are enabled (disableDNS is false).</td>
<td>Optional</td>
</tr>
<tr>
<td>server.id</td>
<td>&lt;string&gt;</td>
<td>This parameter specifies the server id. The server id is used to uniquely identify each container instance. For example, each container gets its own persistent directory based on the server id. By default, the standalone container server id is &quot;&lt;ip&gt;-&lt;containerPort&gt;&quot;. In Tomcat, the server id will default to &quot;&lt;ip&gt;-&lt;webApplicationName&gt;&quot;.</td>
<td>Optional</td>
</tr>
</tbody>
</table>
Table 8.2. Standalone/embedded container-specific configuration parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>container-Timeout</td>
<td>&lt;int&gt;</td>
<td>This parameter controls the container timeout. That is, the maximum amount of</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>time the container will wait to receive a message from the client. By default</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>it is set to 3 minutes.</td>
<td></td>
</tr>
<tr>
<td>webContext</td>
<td>&lt;name&gt;</td>
<td>This parameter specifies the context name under which the services are</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>published under: http://&lt;host&gt;:&lt;port&gt;/&lt;webContext&gt;/services/MyService. By</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>default &quot;wsrf&quot; name is used. In Tomcat, this parameter is set to the web</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>application's name.</td>
<td></td>
</tr>
</tbody>
</table>

2.2. Container Performance Tuning

While there aren't a large number of parameters which you can use to tune the container for performance, configuring your container in a manner appropriate to the needs of your application can yield significant benefits. Having said that, these are very general guidelines which may not apply to every situation. In addition, many commonly used services have configuration recommendations. It is strongly suggested that you check the performance tuning guide for your particular service before attempting to change these settings on your own.

2.2.1. JVM Parameters

The first major parameters you can adjust are not strictly container parameters, but tweaking the parameters of the JVM in which your container is running can provide significant performance benefits. Each implementation of the JVM has its own set of parameters which can be modified, however, all JVM's have parameters to change the heap memory usage. By default, the JVM claims 64 MB from the operating system. This will be too small for any real deployment of the container. For a service such as GRAM, which expects to process large numbers of jobs which may exist for an extended period of time, we recommend specifying at least 1024 MB for the maximum heap memory. In addition, you should specify the minimum amount of heap memory your container will use. To set these values for the container, execute this command:

```
GLOBUS_OPTIONS="-Xms256M -Xmx1024M"
```

This will tell the virtual machine to claim 256 MB as the minimum heap size and 1024 MB as the maximum heap size. As a general rule of thumb, if you specify more memory, your performance will improve. Of course, this is only the case if your machine has sufficient memory.

In addition, certain JVM implementations have additional parameters which can improve performance. The most notable is in the Sun JVM implementation. Specifying the process to be -server allows the JVM to run in "server" mode. The JVM will typically perform better on CPU intensive tasks (like most GT containers) when in "server" mode. To set this value, add -server to your GLOBUS_OPTIONS variable:

```
GLOBUS_OPTIONS="$GLOBUS_OPTIONS -server"
```

2.2.2. Container Parameters

The Container parameters also play a significant role in the performance of your container. The main parameters relate to threads.
Setting up threads for a container is as much an art as a science, so here are some guidelines for configuring your container. The number of threads determines how many concurrent requests your container can service. However, just setting this number to a high value is not a good solution, because each thread takes resources. Ideally, you would want to set this number to be equal to the number of processors your machine has, because that would mean that each thread is being serviced all the time (of course, that is assuming that your computer isn't doing anything else, which is never. That's why this is "ideally"). However, unless you are getting only 1 or 2 requests at a time, this will prevent requests from being processed in a timely manner. So, when configuring this value, you will generally want to find a balance between the number of requests you expect to receive and the resources available on the machine. You should usually try to set this value as low as you can. Having said that, if you expect a large number of requests and those requests are either completed quickly or do the work in a separate thread, the default value of 20 maximum threads is usually sufficient. However, in that scenario, you will probably want to set the minimum number of threads to 20 as well. That way you won't take time starting and stopping threads.

So, as a basic rule of thumb, the default thread configuration will be sufficient unless, you a) expect to receive a large number of concurrent requests which can be processed quickly (in which case you should make the minimum and maximum number of threads equal) or b) you will have a low number of concurrent requests that can be processed quickly (in which case, you might consider reducing the maximum number of threads) or c) you have will receive a large number of concurrent requests which take a long time to process (in which case you should increase the maximum number of threads). The third situation listed is definitely not ideal, because it will put a heavy load on the server with no real way to balance it. In this scenario, you should consider passing the processing from your service to the Work-Manager. The specific configuration parameters are listed below.

### Table 8.3. Container Thread Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>containerThreads</td>
<td>&lt;int&gt;</td>
<td>This parameter controls the initial thread pool size for the container. If not set, it defaults to 1.</td>
<td>Optional</td>
</tr>
<tr>
<td>containerThreadsMax</td>
<td>&lt;int&gt;</td>
<td>This parameter sets the maximum number of threads for the container. By default it is set to 4 * the containerThread setting.</td>
<td>Optional</td>
</tr>
<tr>
<td>containerThreadsHighWaterMark</td>
<td>&lt;int&gt;</td>
<td>This parameter controls when the thread pool of the container should start shrinking (if the number of idle threads exceeds this number). By default it is set to 2 * the containerThread setting.</td>
<td>Optional</td>
</tr>
</tbody>
</table>

The default Thread settings are as follows:

### Table 8.4. Default container thread pool settings

<table>
<thead>
<tr>
<th>Type</th>
<th>Min. threads</th>
<th>Max. threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>standalone</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>embedded</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

### 2.2.3. Client Parameters

The connection timeout parameter in the client can also play a role in tuning performance. This is a new feature in the 4.2 release. It allows a client to keep the http connection open to the server across multiple requests. Setting this to a higher value will mean the same connection is used for longer. This can positively impact performance by eliminating some of the overhead involved in connecting to the server. However, this is not a magic bullet, so don't expect to see huge improvements from this parameter. If you have a long running task on the server, you will probably see modest improvement in the time required to actually submit the request to the server, but the latency in your request will likely
make that mostly moot. Also, if you keep a connection open, you will be limiting the ability of other clients to connect. So this is a parameter that should be changed carefully, depending on your particular circumstances.

2.3. Service Configuration

2.3.1. WSDD

An example of a deployment descriptor for a CounterService:

```xml
<service name="CounterService" provider="Handler" use="literal" style="document">
  <parameter name="className" value="org.globus.wsrf.samples.counter.CounterService"/>
  <parameter name="handlerClass" value="org.globus.axis.providers.RPCProvider"/>
  <parameter name="scope" value="Application"/>
  <wsdlFile>share/schema/core/samples/counter/counter_service.wsdl</wsdlFile>
  <parameter name="allowedMethodsClass" value="com.counter.CounterPortType"/>
  <parameter name="providers" value="DestroyProvider SetTerminationTimeProvider GetRPProvider SubscribeProvider GetCurrentMessageProvider"/>
</service>
```

Services are defined in a `<service>` element. The "name" attribute of the `<service>` element defines the remotely accessible name of the service. The service handle will have the form of `<hosting environment URL>/foo`, where:

- the hosting environment URL typically is http://<host>:<port>/wsrf/services.
- `foo` is the name of the service (`<service name="foo" ...>`).

The `use` attribute should be set to `literal` and the `style` attribute to `document` for all WSRF/WSN based services. The configuration information for a service is defined by various `<parameter>` sub-elements within a `<service>` element. The configuration item `name` corresponds to the "name" attribute in a `<parameter>` sub element, and the `value` is put as a "value" attribute within the same parameter element.
### Table 8.5. Axis Standard Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>className</td>
<td>&lt;class&gt;</td>
<td>This parameter specifies a class that implements the web service methods.</td>
<td>Required</td>
</tr>
<tr>
<td>handler-Class</td>
<td>&lt;class&gt;</td>
<td>This parameter specifies what dispatcher to use to send a request to a service method. This parameter is required if the provider attribute of the service is set to Provider. The default dispatcher we provide is called org.globus.axis.providers.RPCProvider. It enables special features such as operation providers or security support.</td>
<td>Recommended in our environment</td>
</tr>
<tr>
<td>scope</td>
<td>&lt;value&gt;</td>
<td>Scope value can be one of: Request (the default), Application, or Session. If Request scope is used, a new service object is created for each SOAP request that comes in for the service. If Application scope is used, only a single instance of the service object is created and used for all SOAP requests that come in for the service. If Session scope is used, a new service object is created for each session-enabled client who accesses the service. Note: Only Request and Application scopes are supported when used with org.globus.axis.providers.RPCProvider handlerClass.</td>
<td>Application scope is recommended</td>
</tr>
<tr>
<td>wsdlFile</td>
<td>&lt;path&gt;</td>
<td>This parameter points to a wsdl file for the service. The wsdl file must contain the wsdl:service entry. The file location can be relative or absolute. A relative file location is recommended.</td>
<td>Required in our environment</td>
</tr>
<tr>
<td>allowedMethods</td>
<td>&lt;list of methods&gt;</td>
<td>This parameter specifies a space or comma separated list of method names that can be called via SOAP. &quot;*&quot; indicates that all methods of the service class can be invoked via SOAP.</td>
<td>Optional. By default all methods are allowed.</td>
</tr>
</tbody>
</table>

### Table 8.6. Java WS Core Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>loadOnStartup</td>
<td>&lt;boolean&gt;</td>
<td>If set to true this parameter will cause the web service and the corresponding ResourceHome (if any) to be initialized (with proper security settings if configured) at container startup. This is useful for restarting some tasks, etc. at container startup without having to call the service. Please check Section 1.1, “How can I force my service to initialize when the container starts?” for details.</td>
<td>Optional</td>
</tr>
<tr>
<td>allowedMethodsClass</td>
<td>&lt;class&gt;</td>
<td>This parameter is similar to the allowedMethods standard Axis property but it specifies a Java class or an interface that is introspected to come up with a list of allowed methods that can be called remotely on the service. It is useful for easily restricting the SOAP-accessible methods of the service. Usually the class specified in this parameter would be the remote interface class generated for the service. This parameter only has effect if used with org.globus.axis.providers.RPCProvider handlerClass.</td>
<td>Optional</td>
</tr>
<tr>
<td>providers</td>
<td>&lt;list of providers&gt;</td>
<td>This parameter specifies a space separated list of provider names or class names. Please see the Operation provider support section for details. This parameter only has effect if used with org.globus.axis.providers.RPCProvider handlerClass.</td>
<td>Optional</td>
</tr>
</tbody>
</table>
Please see Custom Deployment \(^1\) for details on Axis Web Services Deployment Descriptor.

## 2.3.2. JNDI

An example of a JNDI configuration bit for a CounterService:

```xml
<service name="CounterService"> <resource name="home" type="org.globus.wsrf.samples.counter.CounterHome"> <resourceParams>
<parameter> <name>factory</name> <value>org.globus.wsrf.jndi.BeanFactory</value> </parameter>
<parameter> <name>resourceClass</name> <value>org.globus.wsrf.samples.counter.PersistentCounter</value> </parameter>
<parameter> <name>resourceKeyName</name> <value>{http://counter.com}CounterKey</value> </parameter>
<parameter> <name>resourceKeyType</name> <value>java.lang.Integer</value> </parameter>
</resourceParams> </resource> </service>
```

Each service in WSDD should have a matching entry in the JNDI configuration file with the same name. Under each service entry in JNDI different resource objects or entries might be defined. Please see Section 7, “JNDI, Resources and Objects” for details. Each service entry in JNDI should have a resource defined called "home". That resource is the ResourceHome implementation for the service (as specified by the type attribute). Depending on the ResourceHome implementation different options can be configured for the ResourceHome. Currently we have two main base ResourceHome implementations: org.globus.wsrf.impl.ResourceHomeImpl and org.globus.wsrf.impl.ServiceResourceHome.

*Note:* All "home" resources must specify a factory parameter with org.globus.wsrf.jndi.BeanFactory value.

### 2.3.2.1. ResourceHomeImpl

The ResourceHomeImpl is a generic and reusable ResourceHome implementation. It supports persistent resources, resource caching, resource sweeper, etc.

---

\(^1\) [http://ws.apache.org/axis/java/user-guide.html#PublishingServicesWithAxis](http://ws.apache.org/axis/java/user-guide.html#PublishingServicesWithAxis)
Table 8.7. ResourceHomeImpl parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceKey-Name</td>
<td>&lt;qname&gt;</td>
<td>This parameter specifies a QName of the resource key. The namespace is specified in the <code>{}</code>. For example, this QName will be used to discover the SOAP header that contains the key of the resource in the request.</td>
<td>Required</td>
</tr>
<tr>
<td>resourceKeyType</td>
<td>&lt;class&gt;</td>
<td>This parameter specifies the type of the resource key as a Java class. The key XML element is deserialized into this Java type. The Java type can be for any simple Java type, Axis generated bean, or a class with a type mapping.</td>
<td>Optional. Defaults to java.lang.String</td>
</tr>
<tr>
<td>resourceClass</td>
<td>&lt;class&gt;</td>
<td>This parameter specifies the classname of the resource object. This is used to ensure that right type of resource object is added to resource home and to instantiate the right object if the resource supports persistence.</td>
<td>Required</td>
</tr>
<tr>
<td>sweeperDelay</td>
<td>&lt;long&gt;</td>
<td>This parameter specifies how often the resource sweeper runs in milliseconds.</td>
<td>Optional. Defaults to 1 minute</td>
</tr>
<tr>
<td>cacheLocation</td>
<td>&lt;jndi path&gt;</td>
<td>This parameter specifies the JNDI location of the resource cache for this resource home. Please see Configuring Resource Cache below for details.</td>
<td>Optional</td>
</tr>
</tbody>
</table>

2.3.2.1. Configuring Resource Cache

If ResourceHomeImpl is configured with resource class that implements the PersistenceCallback interface it will store the resource objects wrapped in Java SoftReference². That allows the JVM to automatically reclaim these resource objects, thus reducing the memory usage. Since the JVM can decide to reclaim these objects at any point, sometimes a resource object can be reclaimed between two subsequent invocations on the same resource. This for example can cause the state of the resource to be reloaded from disk on each call. To prevent the JVM from reclaiming the resource objects so quickly a cache can be setup up to hold direct references to these objects. A basic LRU (least recently used) cache implementation is provided. Other cache implementations can be used as long as they implement the org.globus.wsrf.utils.cache.Cache interface.

To configure a cache for ResourceHomeImpl first define a cache resource entry in JNDI, for example:

```xml
<resource name="cache" type="org.globus.wsrf.utils.cache.LRUCache">
    <resourceParams>
        <parameter>
            <name>factory</name>
            <value>org.globus.wsrf.jndi.BeanFactory</value>
        </parameter>
        <parameter>
            <name>timeout</name>
            <value>120000</value>
        </parameter>
        <parameter>
            <name>maxSize</name>
        </parameter>
    </resourceParams>
</resource>
```

² http://java.sun.com/j2se/1.4.2/docs/api/java/lang/ref/SoftReference.html
The "timeout" parameter (in ms) is used to specify the idle time of the resource object before it is removed from the cache. Also, the "maxSize" parameter can be used to specify the maximum size of the cache. By default the cache is configured without any size limit and 5 minutes timeout.

Once the cache resource entry is defined add the "cacheLocation" parameter to the service home resource. The "cacheLocation" parameter value is the JNDI name of the cache resource:

```
<service name="CounterService">
  <resource name="home" type="...">
    <resourceParams>
      ...
      <parameter>
        <name>cacheLocation</name>
        <value>java:comp/env/services/CounterService/cache</value>
      </parameter>
      ...
    </resourceParams>
  </resource>
  ...
  <resource name="cache" type="org.globus.wsrf.utils.cache.LRUCache">
    ...
  </resource>
</service>
```

Please note that once the object is removed from the cache it is still up to the JVM to actually reclaim the object. Also, the same cache resource can be reused in different services but usually one cache per service should be configured.

**2.3.2.2. ServiceResourceHome**

This implementation does not accept any special parameters.

**2.4. Usage Statistics Configuration**

Java WS Core container and other GT services are configured to send out usage statistics. Please see the Usage Statistics section in the Java WS Core Admin Guide for more information.

The targets to which the usage statistics are sent are configured via the usageStatisticsTargets parameter defined in the `<globalConfiguration>` section of the $GLOBUS_LOCATION/etc/globus_wsrf_core/server-config.wsdd file. The usageStatisticsTargets parameter specifies a space separated list of targets to which the usage statistics of various components will be sent to. Each target is of form: host[:port] (port is optional, if not specified a default port will be assumed). By default usage statistics are sent to usage-stats.globus.org:4810.

To disable sending of the usage statistics remove this parameter, comment it out, or remove all of its values.
2.5. Configuration Profiles

Configuration profiles allow for the same Java WS Core installation to have multiple configurations. That is, the same installation can be used to run different containers each with different configuration.

When a .gar file is deployed, a -Dprofile option can be specified to deploy the configuration files under a specific profile name. If the profile name is specified, the deploy operation will drop the configuration file as $GLOBUS_LOCATION/etc/<gar.id>/<profile.name>-server-config.wsdd and/or $GLOBUS_LOCATION/etc/<gar.id>/<profile.name>-jndi-config.xml. The configuration profiles can also be created by hand simply by copying and/or renaming the configuration files appropriately. Each configuration profile should duplicate the contents of $GLOBUS_LOCATION/etc/globus_wsrf_core/server-config.wsdd and $GLOBUS_LOCATION/etc/globus_wsrf_core/jndir-config.xml in order to have the basic functionality work properly.

Once a configuration profile is created, the standalone container can be started with a -profile option to load configuration files in a specific profile.

3. Configuring an /etc/init.d entry for the standalone container

To create an /etc/init.d entry for the standalone container do the following steps:

1. As root create /etc/init.d/gt4container script with the following contents:

   ```
   ACCOUNT=globus GLOBUS_LOCATION=<globusLocation>
   INIT=$GLOBUS_LOCATION/etc/init.d/globus-ws-java-container su $ACCOUNT -c "$INIT $*"
   ```

2. Set executable permissions on the /etc/init.d/gt4container script and register it with the init.d system:

   ```
   $ chmod +x /etc/init.d/gt4container $ /sbin/chkconfig -a gt4container
   ```

   After this step the container should start automatically after the next reboot. Make sure the container is configured with a global security descriptor that explicitly points to long term credentials.

To start the container by hand run:

```
$ /etc/init.d/gt4container start
``` 

To stop the container by hand run:

```
$ /etc/init.d/gt4container stop
```
Chapter 9. Environment variable interface

1. Environmental variables for Java WS Core

Table 9.1. Globus standard environment variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLOBUS_LOCATION</td>
<td>&lt;path&gt;</td>
<td>The &lt;path&gt; is the root location of the Java WS Core installation. Must be an absolute path.</td>
<td>Required</td>
</tr>
<tr>
<td>GLOBUS_TCP_PORT_RANGE</td>
<td>&lt;min,max&gt;</td>
<td>The &lt;min,max&gt; is the minimum and maximum port range for TCP server sockets (useful for systems behind firewalls). For example, if set, the notification sink on the client will be started within that port range.</td>
<td>Optional</td>
</tr>
<tr>
<td>GLOBUS_TCP_SOURCE_PORT_RANGE</td>
<td>&lt;min,max&gt;</td>
<td>The &lt;min,max&gt; is the minimum and maximum port range for TCP outgoing sockets (useful for systems behind firewalls).</td>
<td>Optional</td>
</tr>
<tr>
<td>GLOBUS_UDP_SOURCE_PORT_RANGE</td>
<td>&lt;min,max&gt;</td>
<td>The &lt;min,max&gt; is the minimum and maximum port range for UDP outgoing sockets (useful for systems behind firewalls).</td>
<td>Optional</td>
</tr>
<tr>
<td>GLOBUS_HOSTNAME</td>
<td>&lt;host&gt;</td>
<td>The &lt;host&gt; is either a hostname or ip address. The host ip address under which the container and services will be exposed.</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Table 9.2. Launch script specific environment variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLOBUS_OPTIONS</td>
<td>&lt;arguments&gt;</td>
<td>The &lt;arguments&gt; are arbitrary arguments that can be passed to the JVM. See below for a detailed list of supported options.</td>
<td>Optional</td>
</tr>
<tr>
<td>JAVA_HOME</td>
<td>&lt;path&gt;</td>
<td>The &lt;path&gt; is the root location of the JVM installation. If set, the JVM from that installation will be used. Otherwise, the first one found in path will be used.</td>
<td>Optional</td>
</tr>
<tr>
<td>CLASSPATH</td>
<td>&lt;classpath&gt;</td>
<td>This environment property is ignored by launch scripts.</td>
<td>Ignored</td>
</tr>
</tbody>
</table>
Table 9.3. Options supported by the GLOBUS_OPTIONS environment property

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Dorg.globus.wsrf.proxy.port</td>
<td>int</td>
<td>This property specifies the port number of the proxy server. The proxy server must run on the same machine as the container. This setting will cause the service address to have the port of the proxy instead of the container (only applies to code that uses the ServiceHost or AddressingUtils API).</td>
</tr>
<tr>
<td>-Dorg.globus.wsrf.container.server.id</td>
<td>string</td>
<td>This property specifies the server id. The server id is used to uniquely identify each container instance. For example, each container gets its own persistent directory based on the server id. By default the standalone container will store the persistent resources under the ~/.globus/persisted/&lt;ip&gt;-&lt;containerPort&gt; directory. While in Tomcat the ~/.globus/persisted/&lt;ip&gt;-&lt;webApplicationName&gt; directory will be used instead. This property overwrites the default server id and therefore indirectly controls which storage directory is used by the container. If set, the container will store the persisted resources under ~/.globus/persisted/&lt;server.id&gt;/ instead. Note, that if somehow multiple containers running as the same user on the same machine end up with the same server id / persistent directory they might overwrite each other’s persistent data.</td>
</tr>
<tr>
<td>-Dorg.globus.wsrf.container.persistence.dir</td>
<td>directory</td>
<td>This property specifies the base directory that will be used for storing the persistent resources. This property overwrites the default (~/.globus/persisted/) base directory assumed by the container.</td>
</tr>
</tbody>
</table>

Any JVM options can also be passed using the GLOBUS_OPTIONS environment property.
Chapter 10. Debugging

You can also find information about sys admin logging in Chapter 6, Debugging.

1. Tracing SOAP messages

1.1. Using MessageLoggingHandler

The simplest method for logging SOAP messages is to add the `org.globus.wsrf.handlers.MessageLoggingHandler` to the request or response chain in the `server-config.wsdd` or `client-config.wsdd` files.

For example:

```xml
<requestFlow> ... <handler type="java:org.globus.wsrf.handlers.MessageLoggingHandler"/> ... </requestFlow>
```

Then you must enable logging for this handler class in the appropriate `log4j.properties` files and change the logging level to DEBUG:

```ini
log4j.category.org.globus.wsrf.handlers.MessageLoggingHandler=DEBUG
```

1.2. Enabling logging for Axis classes

Another method for tracing SOAP messages is to enable logging for selected Axis classes. Add the following lines to the appropriate `log4j.properties` files:

```ini
log4j.category.org.apache.client.Call=DEBUG
log4j.category.org.apache.axis.transport.http.HTTPSender=DEBUG # enable the following logger for HTTPS/HTTPG transport handlers
log4j.category.org.globus.axis.axis.transport=DEBUG
```

This will log Axis client side calls and Axis HTTP messages.

1.3. Using TcpMonitor

To trace SOAP messages on the wire you can use TcpMon from Apache Axis. After setting the environment using `$GLOBUS_LOCATION/etc/globus-dev-env.[sh|csh|bat]` run:

```bash
$ java org.apache.axis.utils.tcpmon [listenPort targetHost targetPort]
```

If no arguments are used, you have to fill out these values in the GUI. Make sure to also start the standalone container with the proxy server port option set to the `listenPort` value.

2. Debugging Java process

JVM vendors provide useful tools and troubleshooting guides for debugging Java processes. Please use those guides for debugging your programs, for example:
• Sun JDK 1.5: Trouble-Shooting and Diagnostic Guide

• IBM JDK 5.0: Java Diagnostics Guide

3. Debugging hanged Java process

If a Java process appears to hang, for example in case of the standalone container, the list of deployed services is not shown after a while or all requests to the container time out, requesting the JVM thread dump might help diagnose the problem.

To request JVM thread dump run:

```
kill -QUIT <JVM process id>
```

If this command is successful, the thread dump information should be printed to the standard output or error of the Java process. Therefore, the thread dump information might appear on the console of that process or in a file to which the standard output/error of process is redirected to. Please also note that on certain JVMs the thread dump information might be put in a separate file.

When filing bugs of such nature please always include the JVM thread dump information.

4. Debugging Log4j

If you are having problems with configuring Log4j, you can enable internal Log4j debugging by adding `-Dlog4j.debug=true` option on the `java` command line or passing it via the `GLOBUS_OPTIONS` environment property.

---

2. [http://publib.boulder.ibm.com/infocenter/javasdk/v5r0](http://publib.boulder.ibm.com/infocenter/javasdk/v5r0)
Chapter 11. Troubleshooting

For a list of common errors in GT, see Error Codes.
1. Java WS Core Errors
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failed to acquire notification consumer home instance from registry</td>
<td>Caused by javax.naming.NameNotFoundException: Name services is not bound in this Context error. This warning is logged by the container if the request did not contain the necessary WS-Addressing headers or those headers at all or is somehow misconfigured.</td>
</tr>
<tr>
<td>The WS-Addressing 'To' request header is missing</td>
<td>If you see this error in the container log, it usually means you are trying to connect to HTTPS server using http protocol instead of https. This warning is logged by the container if the request did not contain the necessary WS-Addressing headers or those headers at all or is somehow misconfigured.</td>
</tr>
<tr>
<td>java.io.IOException: Token length X &gt; 33554432</td>
<td>This warning is logged by the container if the token length exceeds 33554432. If you see this error in the container log, it usually means you are trying to connect to HTTPS server using http protocol instead of https.</td>
</tr>
<tr>
<td>java.lang.NoSuchFieldError: DOCUMENT</td>
<td>This error usually indicates a mismatch between the version of Apache Axis that the code was compiled with and the version of Axis that the code is currently running with.</td>
</tr>
<tr>
<td>org.globus.wsrfs.InvalidResourceKeyException: Argument key is null / Resource key is missing</td>
<td>These errors usually indicate that a resource key was not passed with the request or that an invalid resource key was passed with the request (that is, the element QName of the resource key did not match what the service expected).</td>
</tr>
<tr>
<td>Unable to connect to localhost:xxx</td>
<td>Cannot resolve localhost. The machine's /etc/hosts isn't set up correctly and/or you do not have DNS for these machines.</td>
</tr>
<tr>
<td>org.globus.common.ChainedIOException: Failed to initialize security context</td>
<td>This may indicate that the user's proxy is invalid.</td>
</tr>
<tr>
<td>Error: org.xml.sax.SAXException: Unregistered type: class xxx</td>
<td>This may indicate that an Axis generated XML type, defined by the WS RLS XSD, was not properly registered upon deployment without intervention by the user, sometimes they do not.</td>
</tr>
<tr>
<td>No socket factory for 'https' protocol</td>
<td>When a client fails with the following exception: java.io.IOException: No socket factory for 'https' protocol at org.apache.axis.transport.http.HTTPSender.getSocket(HTTPSender.java:179) at org.apache.axis.transport.http.HTTPSender.writeToSocket(HTTPSender.java:397) at org.apache.axis.transport.http.HTTPSender.invoke(HTTPSender.java:135)</td>
</tr>
<tr>
<td>Error Code</td>
<td>Definition</td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
</tr>
</tbody>
</table>
| No client transport named 'https' found | When a client fails with the following exception: 
No client transport named 'https' found at
org.apache.axis.client.AxisClient.invoke(AxisClient.java:170) at
org.apache.axis.client.Call.invokeEngine(Call.java:2726)
The client is most likely loading an incorrect client-config.wsdd configuration file. |
| ConcurrentModificationException in Tomcat 5.0.x | If the following exception is visible in the Tomcat logs at startup, it might cause the HTTPSValve to fail: 
java.util.ConcurrentModificationException at
java.util.HashMap$HashIterator.nextEntry(HashMap.java:782) at
java.util.HashMap$EntryIterator.next(HashMap.java:824) at
java.util.HashMap.putAllForCreate(HashMap.java:424) at
java.util.HashMap.clone(HashMap.java:656) at
mx4j.server.DefaultMBeanRepository.clone(DefaultMBeanRepository.java:56)
The HTTPSValve might fail with the following exception: 
java.lang.NullPointerException at
org.apache.coyote.tomcat5.CoyoteRequest.setAttribute(CoyoteRequest.java:1472) at
org.apache.coyote.tomcat5.CoyoteRequestFacade.setAttribute(CoyoteRequestFacade.java:351) at
org.globus.tomcat.coyote.valves.HTTPSValve.expose(HTTPSValve.java:99)
These exceptions will prevent the transport security from working properly in Tomcat. |
| java.net.SocketException: Invalid argument or cannot assign requested address | FIXME - what causes this? |
| GAR deploy/undeploy fails with container is running error | A GAR file can only be deployed or undeployed locally while the container is off. However, GAR deployment may fail with this error even if the container is off. This usually happens if the container has crashed or was stopped from cleaning up its state files. |

2. General troubleshooting information

- In general, if you want to investigate a problem on your own please see Chapter 10, Debugging for details on how to turn on debugging.
- Most of the command line clients have a -debug option that will display more detailed error messages, including the error stack traces.
- Search the mailing lists\(^1\) such as gt-user@globus.org\(^2\) or jwscore-user@globus.org\(^3\) (before posting a message).
- If you think you have found a bug please report it in our Bugzilla\(^4\) system. Please include as much as detail about the problem as possible.

---

\(^2\) [mailto:gt-user@globus.org](mailto:gt-user@globus.org)
\(^3\) [mailto:jwscore-user@globus.org](mailto:jwscore-user@globus.org)
\(^4\) [http://bugzilla.globus.org/bugzilla/](http://bugzilla.globus.org/bugzilla/)
Chapter 12. Related Documentation

Overview material about WSRF and WSN and more information on the implemented WSDL and schema can be found here\(^1\).

Information about ongoing standards work can be found here:

- WS-Addressing\(^2\)
- WS-ResourceFramework\(^3\)
- WS-Notification\(^4\)

\(^1\) http://www.globus.org/wsrf/
\(^2\) http://www.w3.org/2002/ws/addr/
\(^3\) http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf
\(^4\) http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn
Appendix A. Proxy support

A basic proxy support is provided. The `org.globus.wsrf.proxy.port` system property can be set to the port of the proxy server (the proxy server must run on the same machine as the container). This will make any code that uses the ServiceHost or AddressingUtils API return the address of the proxy server instead of the container. This could be useful, for example, for debugging purposes. The `org.globus.wsrf.proxy.port` system property can be passed to `globus-start-container` script via the GLOBUS_OPTIONS environment property. For example:

```bash
$ setenv GLOBUS_OPTIONS="-Dorg.globus.wsrf.proxy.port=5555" $
.globus-start-container
```

Please note that not all of the code will obey the proxy port setting.
Appendix B. Creating source and binary distributions

The wsrf/build.xml Ant script can be used generate Java WS Core source and/or binary distributions. Use the following syntax to create the distributions:

```
ant <distTask> -Dsrc.tag=<tag> 
    -Dversion=<version>
```

The `<distTask>` must be one of:

- `distSource` - create source code only distribution
- `distBinary` - create binary only distribution
- `dist` - create source and binary distributions

The `-Dsrc.tag` specifies the CVS tag and `-Dversion` specifies the version of the code. Make sure to execute this Ant script from within the `wsrf/` directory. Each of these tasks checks out the source code from the CVS and uses it to create the distributions.

Source-only example:

```
ant distSource -Dsrc.tag=globus_4_0_3 -Dversion=4.0.3
```

The above command will create `ws-core-<version>-src.zip` and `ws-core-<version>-src.tar.gz` files under the `wsrf/` directory.

Binary-only example:

```
ant distBinary -Dsrc.tag=globus_4_0_3 -Dversion=4.0.3
```

The above command will create `ws-core-<version>-bin.zip` and `ws-core-<version>-bin.tar.gz` files under the `tmp/ws-core-<version>/` directory.

Source and binary example:

```
ant dist -Dsrc.tag=globus_4_0_3 -Dversion=4.0.3
```

The above command will generate both distributions at the same time.

1. In-place binary distributions

A binary distribution can also be created from the existing source code by running (in the `wsrf/` directory):

```
ant distbin
```

The above command will create `ws-core-<version>-bin.zip` and `ws-core-<version>-bin.tar.gz` files in the current directory.
## Appendix C. Technology Dependencies Details For Java WS Core

### 1. Dependencies Details

<table>
<thead>
<tr>
<th>Distributed w/ Library</th>
<th>Filename</th>
<th>Version</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Globus WS Addressing</td>
<td>globus-addressing-1.0.jar</td>
<td>Modified from Apache Addressing 1.0</td>
<td><a href="http://www.globus.org/cog/java/">http://www.globus.org/cog/java/</a></td>
</tr>
<tr>
<td>Java CoG Kit FX</td>
<td>cog-axis.jar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Java CoG Kit FX</td>
<td>cog-tomcat.jar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Java CoG Kit</td>
<td>cog-jglobus.jar</td>
<td>1.2</td>
<td><a href="http://jakarta.apache.org/log4j/">http://jakarta.apache.org/log4j/</a></td>
</tr>
<tr>
<td>Java CoG Kit</td>
<td>log4j-1.2.15.jar</td>
<td>1.2.15</td>
<td><a href="http://www.bouncycastle.org/">http://www.bouncycastle.org/</a></td>
</tr>
<tr>
<td>Java CoG Kit</td>
<td>jce-jdk13-131.jar</td>
<td>1.31</td>
<td></td>
</tr>
<tr>
<td>Java CoG Kit</td>
<td>purets.jar 1</td>
<td>0.9b4</td>
<td><a href="http://www.rtfm.com/puretls/">http://www.rtfm.com/puretls/</a></td>
</tr>
<tr>
<td>Java CoG Kit</td>
<td>cryptix32.jar  1</td>
<td>3.2.0</td>
<td><a href="http://www.cryptix.org/">http://www.cryptix.org/</a></td>
</tr>
<tr>
<td>Java CoG Kit</td>
<td>cryptix-asn1.jar 1</td>
<td></td>
<td><a href="http://www.rtfm.com/puretls/">http://www.rtfm.com/puretls/</a></td>
</tr>
<tr>
<td>Java CoG Kit</td>
<td>cryptix.jar</td>
<td></td>
<td><a href="http://www.cryptix.org/">http://www.cryptix.org/</a></td>
</tr>
<tr>
<td>Apache Xerces</td>
<td>xercesImpl-2.7.1.jar</td>
<td>2.7.1</td>
<td><a href="http://xml.apache.org/xerces2-j/">http://xml.apache.org/xerces2-j/</a></td>
</tr>
<tr>
<td>Apache Xerces</td>
<td>xml-apis.jar</td>
<td>2.7.1</td>
<td><a href="http://xml.apache.org/xerces2-j/">http://xml.apache.org/xerces2-j/</a></td>
</tr>
<tr>
<td>Apache Xerces</td>
<td>resolver.jar</td>
<td>2.7.1</td>
<td><a href="http://xml.apache.org/xerces2-j/">http://xml.apache.org/xerces2-j/</a></td>
</tr>
<tr>
<td>Apache XML Security</td>
<td>xmlsec-1.2.1.jar</td>
<td>1.2.1</td>
<td><a href="http://xml.apache.org/security/">http://xml.apache.org/security/</a></td>
</tr>
<tr>
<td>Apache Axis</td>
<td>axis.jar 1</td>
<td>1.5</td>
<td><a href="http://ws.apache.org/axis/">http://ws.apache.org/axis/</a></td>
</tr>
<tr>
<td>Apache Axis</td>
<td>jaxrpc.jar</td>
<td>1.5</td>
<td><a href="http://ws.apache.org/axis/">http://ws.apache.org/axis/</a></td>
</tr>
<tr>
<td>Apache Axis</td>
<td>saaj.jar</td>
<td>1.5</td>
<td><a href="http://ws.apache.org/axis/">http://ws.apache.org/axis/</a></td>
</tr>
<tr>
<td>Apache Commons</td>
<td>commons-discovery-0.2.jar</td>
<td>0.2</td>
<td><a href="http://jakarta.apache.org/commons/discovery/">http://jakarta.apache.org/commons/discovery/</a></td>
</tr>
<tr>
<td>Apache Commons</td>
<td>commons-logging-1.1.jar</td>
<td>1.1</td>
<td><a href="http://jakarta.apache.org/commons/logging/">http://jakarta.apache.org/commons/logging/</a></td>
</tr>
<tr>
<td>WSDL4J</td>
<td>wsdl4j-1.5.1.jar</td>
<td>1.5.1</td>
<td><a href="http://sourceforge.net/projects/wsd4j/">http://sourceforge.net/projects/wsd4j/</a></td>
</tr>
<tr>
<td>Apache Xalan</td>
<td>xalan-2.6.jar</td>
<td>2.6.0</td>
<td><a href="http://xml.apache.org/xalan-j/">http://xml.apache.org/xalan-j/</a></td>
</tr>
<tr>
<td>Apache WS-Security</td>
<td>wss4j.jar 2</td>
<td></td>
<td><a href="http://ws.apache.org/wss4j/">http://ws.apache.org/wss4j/</a></td>
</tr>
<tr>
<td>Apache Directory</td>
<td>naming-core-0.8.jar</td>
<td>0.8</td>
<td><a href="http://directory.apache.org/subprojects/naming/">http://directory.apache.org/subprojects/naming/</a></td>
</tr>
<tr>
<td>Apache Directory</td>
<td>naming-factory-0.8.jar</td>
<td>0.8</td>
<td><a href="http://directory.apache.org/subprojects/naming/">http://directory.apache.org/subprojects/naming/</a></td>
</tr>
<tr>
<td>Apache Directory</td>
<td>naming-java-0.8.jar</td>
<td>0.8</td>
<td><a href="http://directory.apache.org/subprojects/naming/">http://directory.apache.org/subprojects/naming/</a></td>
</tr>
<tr>
<td>Distributed w/ Library</td>
<td>Filename</td>
<td>Version</td>
<td>URL</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------</td>
<td>---------</td>
<td>-----</td>
</tr>
<tr>
<td>Apache Directory</td>
<td>naming-resources-0.8.jar</td>
<td>0.8</td>
<td><a href="http://directory.apache.org/subprojects/naming/">http://directory.apache.org/subprojects/naming/</a></td>
</tr>
<tr>
<td>Apache Tomcat Servlet API</td>
<td>servlet.jar</td>
<td></td>
<td><a href="http://jakarta.apache.org/tomcat/">http://jakarta.apache.org/tomcat/</a></td>
</tr>
<tr>
<td>Apache Commons</td>
<td>commons-beanutils-1.6.1.jar</td>
<td>1.6.1</td>
<td><a href="http://jakarta.apache.org/commons/beanutils/">http://jakarta.apache.org/commons/beanutils/</a></td>
</tr>
<tr>
<td>Apache Commons</td>
<td>commons-cli-2.0.jar</td>
<td>–2.0</td>
<td><a href="http://jakarta.apache.org/commons/cli/">http://jakarta.apache.org/commons/cli/</a></td>
</tr>
<tr>
<td>Apache Commons</td>
<td>commons-collections-3.2.jar</td>
<td>3.2</td>
<td><a href="http://jakarta.apache.org/commons/collections/">http://jakarta.apache.org/commons/collections/</a></td>
</tr>
<tr>
<td>Apache Commons</td>
<td>commons-digester-1.5.jar</td>
<td>1.5</td>
<td><a href="http://jakarta.apache.org/commons/digester/">http://jakarta.apache.org/commons/digester/</a></td>
</tr>
<tr>
<td>Apache Commons</td>
<td>commons-httpclient-3.0.jar</td>
<td>3.0</td>
<td><a href="http://jakarta.apache.org/commons/httpclient/">http://jakarta.apache.org/commons/httpclient/</a></td>
</tr>
<tr>
<td>Apache Commons</td>
<td>commons-codec-1.3.jar</td>
<td>1.3</td>
<td><a href="http://jakarta.apache.org/commons/codecs/">http://jakarta.apache.org/commons/codecs/</a></td>
</tr>
<tr>
<td>util.concurrent library</td>
<td>concurrent.jar</td>
<td>1.3.4</td>
<td><a href="http://gee.cs.oswego.edu/dl/classes/EDU/oswego/cs/dl/util/concurrent/intro.html">http://gee.cs.oswego.edu/dl/classes/EDU/oswego/cs/dl/util/concurrent/intro.html</a></td>
</tr>
<tr>
<td>Timer and Work Manager API</td>
<td>commonj.jar</td>
<td>1.1</td>
<td><a href="http://dev2dev.bea.com/wplatform/commonj/twm.html">http://dev2dev.bea.com/wplatform/commonj/twm.html</a></td>
</tr>
<tr>
<td>OpenSAML</td>
<td>opensaml-1.1.jar</td>
<td>1.1</td>
<td><a href="http://www.opensaml.org/">http://www.opensaml.org/</a></td>
</tr>
</tbody>
</table>

2. Legend

1. Custom-modified version
2. Custom version

3. Source code details

<table>
<thead>
<tr>
<th>File(s)</th>
<th>Repository</th>
<th>Module</th>
<th>Tag</th>
<th>Source tarball</th>
<th>Misc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>cog-jglobus.jar, cog-url.jar</td>
<td>CoG CVS</td>
<td>jglobus</td>
<td>HEAD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cog-axis.jar, cog-tomcat.jar</td>
<td>CoG FX CVS</td>
<td>src/jglobus-fx/gsi</td>
<td>HEAD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>puretls.jar</td>
<td>CoG CVS</td>
<td>puretls-0.9b4</td>
<td>HEAD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cryptix32.jar</td>
<td>CoG CVS</td>
<td>cryptix32</td>
<td>HEAD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cryptix-asn1.jar</td>
<td>CoG CVS</td>
<td>cryptix-asn1</td>
<td>HEAD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>axis.jar</td>
<td>Apache SVN</td>
<td>/webservices/axis/trunk/java</td>
<td>474902</td>
<td>ws-axis.tar.gz</td>
<td>with patch</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>File(s)</th>
<th>Repository</th>
<th>Module</th>
<th>Tag</th>
<th>Source tarball</th>
<th>Misc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>globus-addressing-1.0.jar</td>
<td>GT CVS</td>
<td>wsrf/java/lib-src/ws-addressing</td>
<td></td>
<td>ws-addressing.tar.gz³</td>
<td>with patch⁴</td>
</tr>
<tr>
<td>wss4j.jar</td>
<td>Apache SVN</td>
<td>/webservices/wss4j/tags/gt395</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>commons-cli-2.0.jar</td>
<td>Apache SVN</td>
<td>/jakarta/commons/proper/cli/trunk</td>
<td>412903</td>
<td>commons-cli.tar.gz⁴</td>
<td></td>
</tr>
</tbody>
</table>

### 4. Repositories

<table>
<thead>
<tr>
<th>Repository Name</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoG CVS Root</td>
<td>cvs.globus.org:/homes/dsl/cog/CVS⁹</td>
</tr>
<tr>
<td>CoG FX CVS Root</td>
<td>cvs.cogkit.org:/cvs/cogkit⁷</td>
</tr>
<tr>
<td>GT CVS Root</td>
<td>cvs.globus.org:/home/globdev/CVS/globus-packages⁸</td>
</tr>
<tr>
<td>Apache SVN Root</td>
<td><a href="http://svn.apache.org/repos/asf%E2%81%B9">http://svn.apache.org/repos/asf⁹</a></td>
</tr>
</tbody>
</table>

⁴ [http://viewcvs.globus.org/viewcvs.cgi/?cvsroot=Java+COG](http://viewcvs.globus.org/viewcvs.cgi/?cvsroot=Java+COG)
⁵ [http://www.cogkit.org/viewcvs/viewcvs.cgi/](http://www.cogkit.org/viewcvs/viewcvs.cgi/)
⁶ [http://viewcvs.globus.org/viewcvs.cgi/?cvsroot=Globus+Packages](http://viewcvs.globus.org/viewcvs.cgi/?cvsroot=Globus+Packages)
⁷ [http://svn.apache.org/viewcvs.cgi/](http://svn.apache.org/viewcvs.cgi/)
Glossary

A
Apache Axis The SOAP engine implementation used within the Globus Toolkit. See the Apache Axis website\(^1\) for details.

C
certificate A public key plus information about the certificate owner bound together by the digital signature of a CA. In the case of a CA certificate, the certificate is self signed, i.e. it was signed using its own private key.

G
GAR The GAR (Grid ARchive) file is a single file which contains all the files and information that the container needs to deploy a service. See the Java WS Core Developer's Guide for details.

J
JNDI Java Naming and Directory Interface (JNDI) API are used to access a central transient container registry. The registry is mainly used for discovery of the ResourceHome implementations. However, the registry can also be used store and retrieve arbitrary information. The jndi-config.xml files are used to populate the registry. See the JNDI Tutorial\(^3\) for details.

jndi-config.xml It is an XML-based configuration file used to populate the container registry accessible via the JNDI API. See in the Java WS Core Developer's Guide\] for details.

R
ResourceHome In Java WS Core, resources are managed and discovered via ResourceHome implementations. The ResourceHome implementations can also be responsible for creating new resources, performing operations on a set of resources at a time, etc. ResourceHomes are configured in JNDI and are associated with a particular web service.

S
server-config.wsdd Axis server-side WSDD configuration file. It contains information about the services, the type mappings and various handlers.

\(^1\) http://ws.apache.org/axis/
\(^3\) http://java.sun.com/products/jndi/tutorial/
### W

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Web Services Addressing (WSA)</strong></td>
<td>The WS-Addressing specification defines transport-neutral mechanisms to address web services and messages. Specifically, it defines XML elements to identify web service endpoints and to secure end-to-end endpoint identification in messages. See the W3C WS Addressing Working Group[^14] for details.</td>
</tr>
<tr>
<td><strong>Web Services Deployment Descriptor (WSDD)</strong></td>
<td>An Axis XML-based configuration file.</td>
</tr>
<tr>
<td><strong>Web Services Description Language (WSDL)</strong></td>
<td>WSDL is an XML document for describing Web services. Standardized binding conventions define how to use WSDL in conjunction with SOAP and other messaging substrates. WSDL interfaces can be compiled to generate proxy code that constructs messages and manages communications on behalf of the client application. The proxy automatically maps the XML message structures into native language objects that can be directly manipulated by the application. The proxy frees the developer from having to understand and manipulate XML. See the WSDL 1.1 specification[^15] for details.</td>
</tr>
<tr>
<td><strong>Web Services Interoperability Basic Profile (WS-I Basic Profile)</strong></td>
<td>The WS-I Basic Profile[^17] specification is a set of recommendations on how to use the different web services specifications such as SOAP, WSDL, etc. to maximize interoperability.</td>
</tr>
<tr>
<td><strong>Web Services Notification (WSN)</strong></td>
<td>The WS-Notification family of specifications define a pattern-based approach to allowing Web services to disseminate information to one another. This framework comprises mechanisms for basic notification (WS-Notification), topic-based notification (WS-Topics), and brokered notification (WS-BrokeredNotification). See the OASIS Web Services Notification (WSN) TC[^18] for details.</td>
</tr>
<tr>
<td><strong>Web Services Resource Framework (WSRF)</strong></td>
<td>Web Services Resource Framework (WSRF) is a specification that extends web services for grid applications by giving them the ability to retain state information while at the same time retaining statelessness (using resources). The combination of a web service and a resource is referred to as a WS-Resource. WSRF is a collection of different specifications that manage WS-Resources. This framework comprises mechanisms to describe views on the state (WS-ResourceProperties), to support management of the state through properties associated with the Web service (WS-ResourceLifetime), to describe how these mechanisms are extensible to groups of Web services (WS-ServiceGroup), and to deal with faults (WS-BaseFaults). For more information, go to: <a href="http://www.globus.org/wsrf/">http://www.globus.org/wsrf/</a> and OASIS Web Services Notification (WSRF) TC[^19].</td>
</tr>
</tbody>
</table>

[^15]: [http://www.w3.org/TR/wsd](http://www.w3.org/TR/wsd)
Extensible Markup Language (XML) is standard, flexible, and extensible data format used for web services. See the W3C XML site[^1] for details.

[^1]: http://www.w3.org/XML/
Index

A
adding a new query/topic expression evaluator, 25
AnyContentType
  working with AnyContentType content, 12
API, 45
architecture, 44

C
call object
  using directly, 10
configuring, 103
/etc/init.d for the standalone container, 111
global, 103
globus-update-client-config, 85
Merges client-config.wsdd files from deployed modules into the global client-config.wsdd configuration file, 85
overview, 103
profiles, 111
service, 106
  JNDI, 108
  WSDD, 106
usage statistics, 110
validating configuration files of all services, 86
connection
controlling connection reuse, 33
controlling connection settings, 33
setting connection timeout, 33
containers
globus-reload-container, 87
globus-start-container, 50
globus-start-container-detached, 53
globus-stop-container, 51
globus-stop-container-detached, 54
obtaining endpoint information, 9
reinitializing the standalone container, 87
starting standalone container, 50
starting standalone container detached from controlling TTY, 53
stopping standalone container, 51
stopping standalone container detached from controlling TTY, 54
creating source and binary distributions, 122

D
debugging, 114
  hanged Java process, 115
Java process, 114
  Log4j, 115
  tracing SOAP messages, 114
    Enabling logging for Axis classes, 114
    using MessageLoggingHandler, 114
    Using TcpMonitor, 114
dependencies
details, 123
deploying GARs
generating launcher scripts, 30
document/literal, 38

E
efficiency
endpoint
  obtaining container and service endpoint information, 9
enumeration
retrieving enumeration data, 95
starting an enumeration, 93
stopping an enumeration, 97
ws-enumerate, 95
ws-enumerate-end, 97
ws-enumerate-start, 93
evironment
displaying component version information and validating JVM version, 83
displaying remote component version information, 84
globus-check-environment, 83
globus-check-remote-environment, 84
evironment variables, 112
  Globus standard, 112
  launch script specific, 112
errors, 117
expression evaluator
adding a new query/topic, 25

G
globus-validate-descriptors, 86
Grid Archive (GAR)
creating a GAR file through Ant, 29
deploying a GAR file, 30
  locally, 79
  remotely, 91
  generating launcher scripts, 30
globus-deploy-gar, 79
globus-remote-deploy-gar, 91
lobus-remote-undeplon-gar, 89
globus-undeplon-gar, 81
undeploying a GAR file
  locally, 81
  remotely, 89

H
HTTP
controlling HTTP chunked encoding, 33

J
Java
debugging hanged Java process, 115
downloading Java process, 114
serializing and deserializing Java objects to and from files, 12

L
Log4j
debugging, 115

M
MessageContext
obtaining standard MessageContext properties, 39

N
notifications
setting up and receiving notifications, 13

O
operation providers, 6

P
presentations, 40
programming
client-side
controlling connection reuse, 33
controlling connection settings, 33
controlling HTTP chunked encoding, 33
controlling HTTP protocol version, 34
setting connection timeout, 33
general
adding and retrieving SOAP Attachments, 36
obtaining version information, 36
querying resource properties using XPath, 23
Serializing and deserializing Java objects to and from files, 12
setting up and receiving notifications, 13
using ReferenceParameters, 11
using the call object directly, 10
working with AnyContentType content, 12
service-side
obtaining container and service endpoint information, 9
obtaining service parameters, 38
obtaining standard MessageContext properties, 39
proxy support, 121

R
ReferenceParameters
using, 11
related documentation
ongoing standards work, 120
overview of WSRF/WSN/WSDL, 120
resource
destroying a resource, 55
setting the termination time of a resource, 57
wsrf-destroy, 55
wsrf-set-termination-time, 57
resource properties
deleting a resource property, 69
getting a single resource property from a resource, 61
getting multiple resource properties from a resource, 63
globus-xpath-query, 99
inserting a property into the resource property document of a resource, 65
performing an XPath query on a resource property document, 99
querying resource properties using XPath, 23
querying the resource property document of a resource, 59
updating the property value in the resource property document of a resource, 67
wsrf-delete-property, 69
wsrf-get-properties, 63
wsrf-get-property, 61
wsrf-insert-property, 65
wsrf-query, 59
wsrf-update-property, 67

S
samples
counter, 41
management, 42
security considerations, 4
service
obtaining service parameters, 38
services
obtaining endpoint information, 9
SOAP
adding and retrieving SOAP Attachments, 36
tracing SOAP messages, 114

Enabling logging for Axis classes, 114
using MessageLoggingHandler, 114
Using TcpMonitor, 114
SOAP Encoding, 38
subscription
getting a current message associated with a topic, 71
pausing a subscription, 73
resuming a subscription, 75
subscribing to a topic, 77
wsn-get-current-message, 71
| T                   | testing, 34                  |
|                    | running, 35                  |
|                    | groups, 35                   |
|                    | single, 35                   |
|                    | test results, 36             |
|                    | writing, 34                  |
| troubleshooting    | for developers, 116          |
| tutorials, 40      |                             |

| V                   | validating configuration files of all services, 86 |
| version             | obtaining version information, 36                 |

| W                   | WSDL, 48                                    |
|                     | WSDL/SOAP rules, 37                         |

| X                   | XPath                                       |
|                     | globus-xpath-query, 99                      |
|                     | performing an XPath query on a resource property document, 99 |
|                     | querying resource properties using XPath, 23 |