GT 4.0: Java WS Core

GT 4.0: Java WS Core

Table of Contents

List of Tables

Chapter 1. GT 4.0 Common Runtime Components: Key Concepts

1. Overview

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

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

2. Conceptual Details

2.1. Web Services

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

2.1.1. GT4, Distributed Systems, and Web Services

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

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

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

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

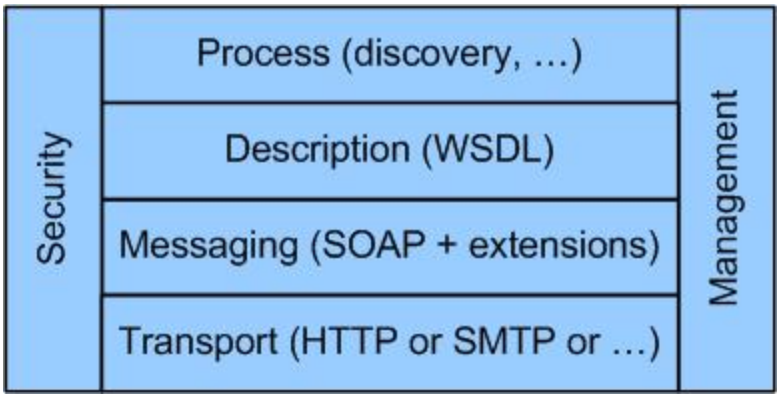

Figure 1: An abstract view of the various specifications that define the Web services architecture

2.1.2. Service Oriented Applications and Infrastructure

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

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

2.1.3. Web Services Implementation

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

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

2. the Web service implementation, the (domain-specific) code that handles the message.

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

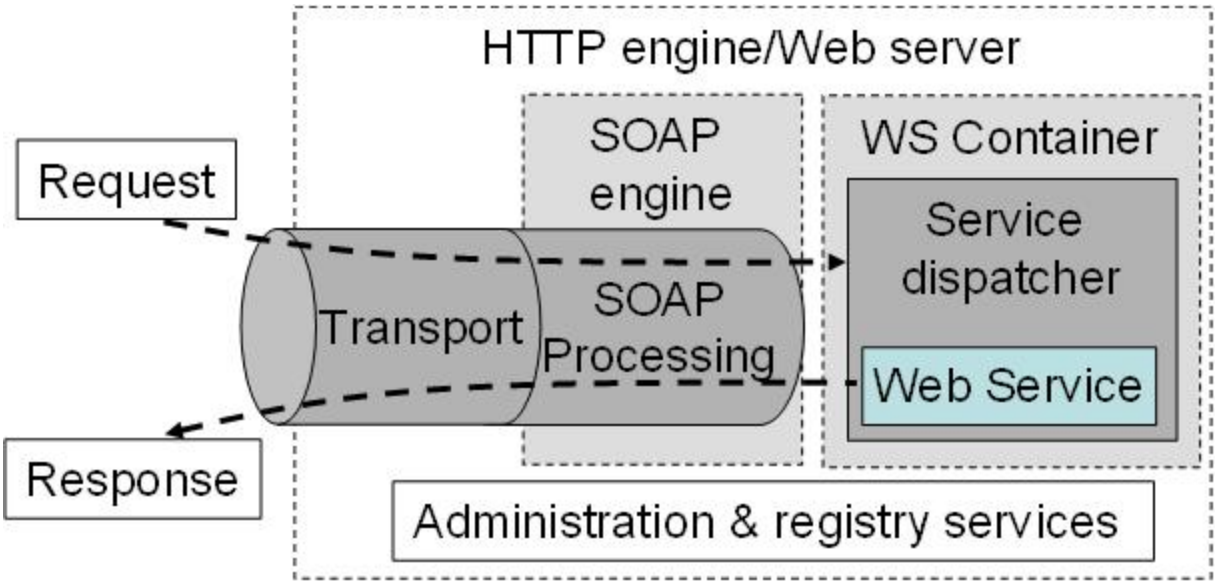

Figure 2: WS Container. High-level picture of functional components commonly encountered in Web service implementations, showing the path taken by requests and responses.

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

2.1.4. Web Services Specifications

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

2.1.5. XML, SOAP, WSDL

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

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

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

2.1.6. WS-Security and Friends

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

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

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

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

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

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

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

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

2.1.8. Other Relevant Specifications

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

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

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

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

2.1.9. Service Oriented Architecture

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

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

- Logical view: The service is an abstracted, logical view of actual programs, databases, business processes, etc., defined in terms of what it does, typically carrying out a business-level operation.

- Message orientation: The service is formally defined in terms of the messages exchanged between provider agents and requester agents, and not the properties of the agents themselves. The internal structure of an agent, including features such as its implementation language, process structure and even database structure, are deliberately abstracted away in the SOA: using the SOA discipline one does not and should not need to know how an agent implementing a service is constructed. A key benefit of this concerns so-called legacy systems. By avoiding any knowledge of the internal structure of an agent, one can incorporate any software component or application that can be "wrapped" in message handling code that allows it to adhere to the formal service definition.

- Description orientation: A service is described by machine-processable metadata. The description supports the public nature of the SOA: only those details that are exposed to the public and important for the use of the service should be included in the description. The semantics of a service should be documented, either directly or indirectly, by its description.

- Granularity: Services tend to use a small number of operations with relatively large and complex messages.

- Network orientation: Services tend to be oriented toward use over a network, though this is not an absolute requirement.

- Platform neutral: Messages are sent in a platform-neutral, standardized format delivered through the interfaces. XML is the most obvious format that meets this constraint.

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

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

3. Related Documents

3.1. Web Services

- Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C. and Orchard, D. Web Services Architecture. W3C, Working Draft[1]

[1] http://www.w3.org/TR/2003/WD-ws-arch-20030808/

Chapter 2. GT 4.0 Release Notes: for Java WS Core

1. Component Overview

The Java WS Core is an implementation of the *Web Services Resource Framework (WSRF)* and the *Web Service Notification (WSN)* family of standards. It provides APIs and tools for building stateful Web services.

2. Feature Summary

New Features in the GT 4.0 release

- Implementation of the 2004/06 OASIS *WSRF* and *WSN* working draft specifications (with minor fixes to the 1.2-draft-01 published schemas and with the March 2004 version of the *WS-Addressing* specification)
- Basic HTTP/1.1 client & server support
- *JNDI*-based registry based on the JNDI service in Apache Tomcat
- An implementation of the Work Manager[1] and Timer[2] specifications

Other Supported Features

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

Deprecated Features

- None

3. Changes Summary

The following changes have occurred for Java WS Core:

3.1. Parameter ordering for Axis generated files

The Java WS Core 4.0 release contains a newer version of Axis. In this version of Axis the ordering of parameters in the constructors of generated types has changed. The parameters are now sorted alphabetically. Code that creates an

[1] http://dev2dev.bea.com/wlplatform/commonj/twm.csp
[2] http://dev2dev.bea.com/wlplatform/commonj/twm.csp

instance of some generated type using a constructor with multiple arguments might need to be checked and updated appropriately. This change does not affect code that creates an instance of some generated type using a default constructor and sets the values using the individual setter methods.

3.2. Transport security is used by default (since 3.9.4)

Transport security (HTTPS) is now assumed as the default security mechanism. For example, the standalone container will now start in a transport security mode (HTTPS container running on port 8443). The plain (HTTP) container can still be started using the `-nosec` option. Please see the globus-start-container(1) documentation for details.

3.3. Different naming scheme for Axis generated files (since 3.9.4)

The Axis version distributed with Java WS Core follows the JAX-RPC specification and Java naming standards more closely than before when naming the generated files (generated by the wsdl2java process). The code generated is exactly the same as before but the names of the files are slightly different. The changes to the names are pretty straightforward: All underscores are dropped, and the first letter of each word within the name is capitalized. If there are collisions between names (for example, the port type name is the same as some element name), the name for the port type will end with `_PortType` and for the element with `_Element`. Examples:

```
foo.java -> Foo.java
_Foo_Bar.java -> FooBar.java
_GetMultipleResourceProperties.java -> GetMultipleResourceProperties_Element.java
GetMultipleResourceProperties.java -> GetMultipleResourceProperties_PortType.java
```

4. Bug Fixes

- Bug 2650[3]
- Bug 2651[4]
- Bug 2664[5]
- Bug 2765[6]
- Bug 2814[7]
- Bug 2828[8]
- Bug 2854[9]
- Bug 2885[10]
- Bug 2887[11]

[3] http://bugzilla.globus.org/globus/show_bug.cgi?id=2650
[4] http://bugzilla.globus.org/globus/show_bug.cgi?id=2651
[5] http://bugzilla.globus.org/globus/show_bug.cgi?id=2664
[6] http://bugzilla.globus.org/globus/show_bug.cgi?id=2765
[7] http://bugzilla.globus.org/globus/show_bug.cgi?id=2814
[8] http://bugzilla.globus.org/globus/show_bug.cgi?id=2828
[9] http://bugzilla.globus.org/globus/show_bug.cgi?id=2854
[10] http://bugzilla.globus.org/globus/show_bug.cgi?id=2885
[11] http://bugzilla.globus.org/globus/show_bug.cgi?id=2887

- Bug 2923[12]
- Bug 2924[13]
- Bug 3080[14]
- Bug 3162[15]
- Bug 3200[16]

5. Known Problems

The following problems and limitations are known to exist for Java WS Core at the time of the 4.0.0 release:

5.1. Limitations

- WS-Notification support:
  - Only the Simple topic dialect is supported (others can be added)
  - Only flat topic spaces are supported (architecture does allow for more advanced structures)
  - Actions on the precondition, selector and policy fields in a subscription are not supported
  - When a resource is removed its subscriptions are not removed automatically
- Only XPath resource property queries are supported (others can be added)
- A resource might not get destroyed at the exact time as indicated by the scheduled termination time. A sweeper thread that removes expired resources runs periodically (every 1 minute by default) so an expired resource might not get removed until the next time the sweeper thread runs.
- SOAP messages with attachments are not supported. In fact, the Axis version distributed with GT was compiled without attachment support.

5.2. Known Bugs

- Bug 2471:[17] Message security signature verification issues
- Bug 2445:[18] Same input and output messages in WSDL confuse Axis
- Bug 2921:[19] Support for TerminationTimeChangeRejectedFault
- Bug 2926:[20] Local transport does not work without a current MessageContext

[12] http://bugzilla.globus.org/globus/show_bug.cgi?id=2923
[13] http://bugzilla.globus.org/globus/show_bug.cgi?id=2924
[14] http://bugzilla.globus.org/globus/show_bug.cgi?id=3080
[15] http://bugzilla.globus.org/globus/show_bug.cgi?id=3162
[16] http://bugzilla.globus.org/globus/show_bug.cgi?id=3200
[17] http://bugzilla.globus.org/globus/show_bug.cgi?id=2471
[18] http://bugzilla.globus.org/globus/show_bug.cgi?id=2445
[19] http://bugzilla.globus.org/globus/show_bug.cgi?id=2921
[20] http://bugzilla.globus.org/globus/show_bug.cgi?id=2926

- Bug 3113:[21] Processing by the WSDLPreprocessor produces output different depending on the JVM
- Bug 3482:[22] wsa:From is not set correctly when service calls another service
- Bug 3483:[23] xsd:anyType not serialized correctly

6. Technology Dependencies

Java WS Core depends on the following GT components:

- Java CoG Kit[24]

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

- Apache Xerces[25]
- Apache XML Security[26]
- Apache Axis[27]
- Apache Xalan[28]
- Apache Addressing[29]
- Apache Commons BeanUtils[30]
- Apache Commons CLI[31]
- Apache Commons Collections[32]
- Apache Commons Digester[33]
- Concurrent Library[34]
- Apache Tomcat JNDI[35]

Please see the Technology Dependencies Details page[36] for details.

7. Tested Platforms

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

[21] http://bugzilla.globus.org/globus/show_bug.cgi?id=3113
[22] http://bugzilla.globus.org/globus/show_bug.cgi?id=3482
[23] http://bugzilla.globus.org/globus/show_bug.cgi?id=3483
[24] http://www.globus.org/cog/java/
[25] http://xml.apache.org/xerces2-j/
[26] http://xml.apache.org/security/
[27] http://ws.apache.org/axis/
[28] http://xml.apache.org/xalan-j/
[29] http://ws.apache.org/ws-fx/addressing/
[30] http://jakarta.apache.org/commons/beanutils/
[31] http://jakarta.apache.org/commons/cli/
[32] http://jakarta.apache.org/commons/collections/
[33] http://jakarta.apache.org/commons/digester/
[34] http://gee.cs.oswego.edu/dl/classes/EDU/oswego/cs/dl/util/concurrent/intro.html
[35] http://cvs.apache.org/viewcvs.cgi/jakarta-tomcat-catalina/catalina/src/share/org/apache/naming/
[36] dependencies.html

Tested platforms for Java WS Core:

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

Tested JVMs for Java WS Core:

- Sun JVM[37] 1.3.1, 1.4.2 and 1.5.0
- IBM JVM[38] 1.3.1, 1.4.1, and 1.4.2
- BEA JRockit JVM[39] 1.5.0

JVM notes:

- GCJ[40] is not supported.
- If using IBM JVM 1.4.1 please see bug 2828[41] for more information.

Tested containers for Java WS Core:

- Java WS Core container
- Tomcat 4.1.31
- Tomcat 5.0.30

8. Backward Compatibility Summary

Protocol changes since GT version 3.2

- HTTP/1.1 with 'chunked' transfer encoding is now used by default.
- Wire messages follow the new schemas and therefore are completely different (see below).

API changes since GT version 3.2

- The majority of the APIs are new. Some APIs resemble GT 3.2 APIs, for example `ServiceData` is replaced by `ResourceProperty` and `ServiceDataSet` is replaced by `ResourcePropertySet`.

Schema changes since GT version 3.2

- Schemas are completely new. The WS Java Core implements the OASIS WSRF and WSN working drafts specifications (with minor fixes to the 1.2-draft-01 published schemas and with the March 2004 version of the WS-Addressing specification.)

[37] http://java.sun.com/j2se/
[38] http://www-106.ibm.com/developerworks/java/jdk/
[39] http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/jrockit
[40] http://gcc.gnu.org/java/
[41] http://bugzilla.globus.org/globus/show_bug.cgi?id=2828#c4

9. For More Information

Please see Java WS Core documentation[42] for more information.

[42] index.html

Chapter 3. GT 4.0.1 Incremental Release Notes: Java WS Core

1. Introduction

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

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

2. Changes Summary

The following changes have occurred for Java WS Core:

- `globus-deploy-gar` now supports `-backup` option to create a backup of existing service configuration files during deployment.
- The standalone container when stopped by pressing `Ctrl-C` will now perform the same cleanup operation as when stopped using the `globus-stop-container` tool.
- The local UDP ports used for sending out the usage statistics information in Java can now be controlled via the `GLOBUS_UDP_SOURCE_PORT_RANGE` environment variable.
- Axis notification handling was improved to use a thread pool for firing notifications instead of starting a new thread for each notification. Also, the memory overhead associated with each notification was reduced.
- The standalone container will now use `~/.globus/persisted/<ip>-<port>/` directory to store its persistent information. Under Tomcat the `~/.globus/persisted/<ip>-<webapp.name>/` directory will be used instead. This change enables to easily run multiple standalone containers as the same user on the same machine without any conflicts or to have multiple deployments of Java WS Core (as different web applications) in the same Tomcat installation.
- Improved Tomcat deployment and support. Also, `ContainerConfig.getBaseDirectory()` and `ContainerConfig.getSchemaDirectory()` API were defined to return the appropriate directory locations depending on the container type. Also, auto flush functionality was added to the Tomcat HTTPS connector code to enable interoperability between a C client and Tomcat 5.0.x.
- Proper WS-Addressing headers are now added to the SOAP Fault messages. Please see Bug 3614[2] for details.

3. Bug Fixes

The following bugs were fixed for Java WS Core:

- Bug 2968:[3] Globalization: Error in JUnit execution in the Japanese locale

[1] http://www.globus.org/toolkit/docs/4.0/common/javawscore/Java_WS_Core_Release_Notes.html
[2] http://bugzilla.globus.org/globus/show_bug.cgi?id=3614
[3] http://bugzilla.globus.org/globus/show_bug.cgi?id=2968

- Bug 3216:[4] Tomcat deployment: copy container-log4j.properties file instead of log4j.properties
- Bug 3217:[5] Get the appropriate base directory location (in Tomcat or in the standalone container) without the need of MessageContext
- Bug 3218:[6] Tomcat deployment: copy libexec/ directory during the deployment
- Bug 3250:[7] ConcurrentModificationException during subscription
- Bug 3257:[8] Improvements to the globus-start-container-detached tool
- Bug 3302:[9] ReflectionResourceProperty inconsistent handling on remove on Array type
- Bug 3418:[10] Support for Ant 1.5.1
- Bug 3466:[11] Error creating persistent directories
- Bug 3472:[12] Secure Conversation Context creation is not thread safe
- Bug 3473:[13] ServiceAuthorizationChain bug in setPolicy node import missing
- Bug 3492:[14] org.globus.wsrf.tools.wsdl.GenerateBinding should use canonical paths for both the bindingFile and serviceFile
- Bug 3532:[15] Recovery thread dies in Tomcat
- Bug 3545:[16] NPE in usage statistics code breaks container
- Bug 3582:[17] globus-stop-container-detached exits with non-0 on successful exit
- Bug 3600:[18] ServerHost.getBaseURL() sometimes returns a wrong URL
- Bug 3614:[19] Missing WS-Addressing headers in fault messages and fault logging problems

4. Known Problems

The following problems and limitations are known to exist for Java WS Core at the time of the 4.0.1 release:

- Limitations
  - WS-Notification support:

[4] http://bugzilla.globus.org/globus/show_bug.cgi?id=3216
[5] http://bugzilla.globus.org/globus/show_bug.cgi?id=3217
[6] http://bugzilla.globus.org/globus/show_bug.cgi?id=3218
[7] http://bugzilla.globus.org/globus/show_bug.cgi?id=3250
[8] http://bugzilla.globus.org/globus/show_bug.cgi?id=3257
[9] http://bugzilla.globus.org/globus/show_bug.cgi?id=3302
[10] http://bugzilla.globus.org/globus/show_bug.cgi?id=3418
[11] http://bugzilla.globus.org/globus/show_bug.cgi?id=3466
[12] http://bugzilla.globus.org/globus/show_bug.cgi?id=3472
[13] http://bugzilla.globus.org/globus/show_bug.cgi?id=3473
[14] http://bugzilla.globus.org/globus/show_bug.cgi?id=3492
[15] http://bugzilla.globus.org/globus/show_bug.cgi?id=3532
[16] http://bugzilla.globus.org/globus/show_bug.cgi?id=3545
[17] http://bugzilla.globus.org/globus/show_bug.cgi?id=3582
[18] http://bugzilla.globus.org/globus/show_bug.cgi?id=3600
[19] http://bugzilla.globus.org/globus/show_bug.cgi?id=3614

    - Only the Simple topic dialect is supported (others can be added)
    - Only flat topic spaces are supported (architecture does allow for more advanced structures)
    - Actions on the precondition, selector and policy fields in a subscription are not supported
    - When a resource is removed its subscriptions are not removed automatically
  - Only XPath resource property queries are supported (others can be added)
  - A resource might not get destroyed at the exact time as indicated by the scheduled termination time. A sweeper thread that removes expired resources runs periodically (every 1 minute by default) so an expired resource might not get removed until the next time the sweeper thread runs.
  - SOAP messages with attachments are not supported. In fact, the Axis version distributed with GT was compiled without attachment support.
  - In certain cases, the "`dialect`" attribute of TopicExpression or QueryExpression is not serialized properly as defined in the schema. An "`org.globus.dialect.attr.qualified`" Java system property was added to control how the serialization of the `dialect` attribute. Please see the Bug 3513[20] for details.
- Known Bugs
  - Bug 2471:[21] Message security signature verification issues
  - Bug 2445:[22] Same input and output messages in WSDL confuse Axis
  - Bug 2921:[23] Support for TerminationTimeChangeRejectedFault
  - Bug 2926:[24] Local transport does not work without a current MessageContext
  - Bug 3113:[25] Processing by the WSDLPreprocessor produces output different depending on the JVM
  - Bug 3482:[26] wsa:From is not set correctly when service calls another service
  - Bug 3483:[27] xsd:anyType not serialized correctly

5. For More Information

Click here[28] for more information about this component.

[20] http://bugzilla.globus.org/globus/show_bug.cgi?id=3513
[21] http://bugzilla.globus.org/globus/show_bug.cgi?id=2471
[22] http://bugzilla.globus.org/globus/show_bug.cgi?id=2445
[23] http://bugzilla.globus.org/globus/show_bug.cgi?id=2921
[24] http://bugzilla.globus.org/globus/show_bug.cgi?id=2926
[25] http://bugzilla.globus.org/globus/show_bug.cgi?id=3113
[26] http://bugzilla.globus.org/globus/show_bug.cgi?id=3482
[27] http://bugzilla.globus.org/globus/show_bug.cgi?id=3483
[28] index.html

Chapter 4. GT 4.0.2 Incremental Release Notes: Java WS Core

1. Introduction

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

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

2. Changes Summary

The following changes have occurred for Java WS Core:

- Added Tomcat 5.5.x support.
- Added TEST.pl for unit tests to integrate with Globus testing framework.
- Fixed additional cases where WS-Addressing headers were not added to the SOAP Fault messages.

3. Bug Fixes

The following bugs were fixed for Java WS Core:

- Bug 3651:[2] Misspelled Deserialization in exception message
- Bug 3737:[3] Support for socket timeouts on the server side
- Bug 3750:[4] Allow for setting the location where the persistence files are stored
- Bug 3259:[5] Generated Unix scripts cannot handle certain variables with spaces
- Bug 3776:[6] globus-start-container requires user to delete orphaned pid file
- Bug 3899:[7] globus-stop-container-detached does not give the container the chance to finish normally
- Bug 3813:[8] Race condition in container creation/shutdown
- Bug 3814:[9] Problem shutting down the container from within one of its threads

[1] http://www.globus.org/toolkit/docs/4.0/common/javawscore/Java_WS_Core_Release_Notes.html
[2] http://bugzilla.globus.org/globus/show_bug.cgi?id=3651
[3] http://bugzilla.globus.org/globus/show_bug.cgi?id=3737
[4] http://bugzilla.globus.org/globus/show_bug.cgi?id=3750
[5] http://bugzilla.globus.org/globus/show_bug.cgi?id=3259
[6] http://bugzilla.globus.org/globus/show_bug.cgi?id=3776
[7] http://bugzilla.globus.org/globus/show_bug.cgi?id=3899
[8] http://bugzilla.globus.org/globus/show_bug.cgi?id=3813
[9] http://bugzilla.globus.org/globus/show_bug.cgi?id=3814

- Bug 3863:[10] ResourcePropertyTopic sendOldValue is ignored
- Bug 3932:[11] Problems with white space when passing options to Windows versions of command wrappers
- Bug 3889:[12] Newline missing when using core serialization utility
- Bug 4107:[13] Notification deadlock
- Bug 4188:[14] Container fails on expired proxy and cannot recover
- Bug 4325:[15] Exceptions of type "expectedType" could be more useful

4. Known Problems

The following problems and limitations are known to exist for Java WS Core at the time of the 4.0.2 release:

- Limitations
  - WS-Notification support:
    - Only the Simple topic dialect is supported (others can be added)
    - Only flat topic spaces are supported (architecture does allow for more advanced structures)
    - Actions on the precondition, selector and policy fields in a subscription are not supported
    - When a resource is removed its subscriptions are not removed automatically
  - Only XPath resource property queries are supported (others can be added)
  - A resource might not get destroyed at the exact time as indicated by the scheduled termination time. A sweeper thread that removes expired resources runs periodically (every 1 minute by default) so an expired resource might not get removed until the next time the sweeper thread runs.
  - SOAP messages with attachments are not supported. In fact, the Axis version distributed with GT was compiled without attachment support.
  - In certain cases, the "`dialect`" attribute of TopicExpression or QueryExpression is not serialized properly as defined in the schema. An "`org.globus.dialect.attr.qualified`" Java system property was added to control how the serialization of the `dialect` attribute. Please see the Bug 3513[16] for details.
- Known Bugs
  - Bug 2471:[17] Message security signature verification issues
  - Bug 2445:[18] Same input and output messages in WSDL confuse Axis

[10] http://bugzilla.globus.org/globus/show_bug.cgi?id=3863
[11] http://bugzilla.globus.org/globus/show_bug.cgi?id=3932
[12] http://bugzilla.globus.org/globus/show_bug.cgi?id=3889
[13] http://bugzilla.globus.org/globus/show_bug.cgi?id=4107
[14] http://bugzilla.globus.org/globus/show_bug.cgi?id=4188
[15] http://bugzilla.globus.org/globus/show_bug.cgi?id=4325
[16] http://bugzilla.globus.org/globus/show_bug.cgi?id=3513
[17] http://bugzilla.globus.org/globus/show_bug.cgi?id=2471
[18] http://bugzilla.globus.org/globus/show_bug.cgi?id=2445

- Bug 2921:[19] Support for TerminationTimeChangeRejectedFault
- Bug 2926:[20] Local transport does not work without a current MessageContext
- Bug 3113:[21] Processing by the WSDLPreprocessor produces output different depending on the JVM
- Bug 3482:[22] wsa:From is not set correctly when service calls another service
- Bug 3483:[23] xsd:anyType not serialized correctly

5. For More Information

Click here[24] for more information about this component.

[19] http://bugzilla.globus.org/globus/show_bug.cgi?id=2921
[20] http://bugzilla.globus.org/globus/show_bug.cgi?id=2926
[21] http://bugzilla.globus.org/globus/show_bug.cgi?id=3113
[22] http://bugzilla.globus.org/globus/show_bug.cgi?id=3482
[23] http://bugzilla.globus.org/globus/show_bug.cgi?id=3483
[24] index.html

Chapter 5. GT 4.0.3 Incremental Release Notes: Java WS Core

1. Introduction

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

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

2. Changes Summary

The following changes have occurred for Java WS Core:

- Adjusted the default size of the thread pool used by the standalone and embedded container. By default the standalone container will now have a minimum of 2 threads and a maximum of 20. The embedded container will now have a minimum of 1 thread and maximum of 3 threads.
- Refreshed version of CoG JGlobus library. Please see the CoG JGlobus Release Notes[2] for more details.
- Updated version of Apache Commons CLI 2.0 library.
- Refreshed version of Apache Axis library with the following bug fixes:
  - Fault bean generation with optional primitive types
  - Rare synchronization problem in symbol table
- Added more TESTS.pl scripts for interop and unit tests to integrate better with the Globus testing framework.
- Tested with Java SE 6 (Beta 2).
- Added two new usage statistics reported by Java WS Core: container uptime and a list of activated services.

3. Bug Fixes

The following bugs were fixed for Java WS Core:

- Bug 4204:[3] WorkManagerImpl incorrectly implements the WorkManager.schedule(...) method
- Bug 4595:[4] wsrf-query with a bad XPath query returns "null"
- Bug 4325:[5] Exceptions of type 'expectedType' could be more useful

[1] http://www.globus.org/toolkit/docs/4.0/common/javawscore/Java_WS_Core_Release_Notes.html
[2] http://www.globus.org/toolkit/docs/4.0/contributions/javacog/JavaCoG_Release_Notes_403.html
[3] http://bugzilla.globus.org/globus/show_bug.cgi?id=4204
[4] http://bugzilla.globus.org/globus/show_bug.cgi?id=4595
[5] http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4325

- Bug 4324:[6] ServiceContainer reports failed startup at incorrect severity
- Bug 4540:[7] globus-build-service.sh and Math example service error

4. Known Problems

The following problems and limitations are known to exist for Java WS Core at the time of the 4.0.3 release:

- Limitations
  - WS-Notification support:
    - Only the Simple topic dialect is supported (others can be added)
    - Only flat topic spaces are supported (architecture does allow for more advanced structures)
    - Actions on the precondition, selector and policy fields in a subscription are not supported
    - When a resource is removed its subscriptions are not removed automatically
  - Only XPath resource property queries are supported (others can be added)
  - A resource might not get destroyed at the exact time as indicated by the scheduled termination time. A sweeper thread that removes expired resources runs periodically (every 1 minute by default) so an expired resource might not get removed until the next time the sweeper thread runs.
  - SOAP messages with attachments are not supported. In fact, the Axis version distributed with GT was compiled without attachment support.
  - In certain cases, the "`dialect`" attribute of TopicExpression or QueryExpression is not serialized properly as defined in the schema. An "`org.globus.dialect.attr.qualified`" Java system property was added to control how the serialization of the `dialect` attribute. Please see the Bug 3513[8] for details.
- Known Bugs
  - Bug 2471:[9] Message security signature verification issues
  - Bug 2445:[10] Same input and output messages in WSDL confuse Axis
  - Bug 2921:[11] Support for TerminationTimeChangeRejectedFault
  - Bug 2926:[12] Local transport does not work without a current MessageContext
  - Bug 3113:[13] Processing by the WSDLPreprocessor produces output different depending on the JVM
  - Bug 3482:[14] wsa:From is not set correctly when service calls another service

[6] http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4324
[7] http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4540
[8] http://bugzilla.globus.org/globus/show_bug.cgi?id=3513
[9] http://bugzilla.globus.org/globus/show_bug.cgi?id=2471
[10] http://bugzilla.globus.org/globus/show_bug.cgi?id=2445
[11] http://bugzilla.globus.org/globus/show_bug.cgi?id=2921
[12] http://bugzilla.globus.org/globus/show_bug.cgi?id=2926
[13] http://bugzilla.globus.org/globus/show_bug.cgi?id=3113
[14] http://bugzilla.globus.org/globus/show_bug.cgi?id=3482

- Bug 3483:[15] xsd:anyType not serialized correctly
- Bug 4432:[16] SimpleTopic.notify(SOAPElement element) drop child elements
- Bug 4566:[17] globus-deploy-gar insists that ANT_HOME be set
- Bug 5147:[18] Unable to create javadocs on JWS Core 4.0.3

5. For More Information

Click here[19] for more information about this component.

[15] http://bugzilla.globus.org/globus/show_bug.cgi?id=3483
[16] http://bugzilla.globus.org/globus/show_bug.cgi?id=4432
[17] http://bugzilla.globus.org/globus/show_bug.cgi?id=4566
[18] http://bugzilla.globus.org/globus/show_bug.cgi?id=5147
[19] index.html

Chapter 6. GT 4.0.4 Incremental Release Notes: Java WS Core

1. Introduction

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

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

2. Changes Summary

- Refreshed version of CoG JGlobus library. Please see the CoG JGlobus Release Notes[2] for more details.
- Refreshed version of Apache Axis library with the following bug fixes:
  - HTTP connections used for one-way requests were not always closed.
  - TypeMapping was not thread-safe (bug 4858[3]).
- Tested with Java SE 6 (RC).

3. Bug Fixes

- Bug 4720:[4] NotificationConsumerManager issues
- Bug 4721:[5] misconfigured wsrf-query script?
- Bug 4560:[6] Changing default ports with ManagedJobFactoryClientHelper

4. Known Problems

- Limitations
  - WS-Notification support:
    - Only the Simple topic dialect is supported (others can be added)
    - Only flat topic spaces are supported (architecture does allow for more advanced structures)
    - Actions on the precondition, selector and policy fields in a subscription are not supported

[1] http://www.globus.org/toolkit/docs/4.0/common/javawscore/Java_WS_Core_Release_Notes.html
[2] http://www.globus.org/toolkit/docs/4.0/contributions/javacog/JavaCoG_Release_Notes_404.html
[3] http://bugzilla.globus.org/globus/show_bug.cgi?id=4858
[4] http://bugzilla.globus.org/globus/show_bug.cgi?id=4720
[5] http://bugzilla.globus.org/globus/show_bug.cgi?id=4721
[6] http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4560

    - When a resource is removed its subscriptions are not removed automatically
  - Only XPath resource property queries are supported (others can be added)
  - A resource might not get destroyed at the exact time as indicated by the scheduled termination time. A sweeper thread that removes expired resources runs periodically (every 1 minute by default) so an expired resource might not get removed until the next time the sweeper thread runs.
  - SOAP messages with attachments are not supported. In fact, the Axis version distributed with GT was compiled without attachment support.
  - In certain cases, the "`dialect`" attribute of TopicExpression or QueryExpression is not serialized properly as defined in the schema. An "`org.globus.dialect.attr.qualified`" Java system property was added to control how the serialization of the `dialect` attribute. Please see the Bug 3513[7] for details.
  - The saml protocol namespace, urn:oasis:names:tc:SAML:1.0:protocol is included in the default list of namespaces to exclude during stub generation. Since core does not use all elements of this namespace, stubs are not generated for all elements in the namespace. Since this namespace is in the default excludes list, stubs will not be generated if the target is used with defaul values for ns.excludes. This has been fixed since this release, but to override this behaviour, set the ns.excludes property appropriately for the generateStubs target. Setting that property to be an empty string will result in stubs being generated for all namespaces.
- Known Bugs
  - Bug 2471:[8] Message security signature verification issues
  - Bug 2445:[9] Same input and output messages in WSDL confuse Axis
  - Bug 2921:[10] Support for TerminationTimeChangeRejectedFault
  - Bug 2926:[11] Local transport does not work without a current MessageContext
  - Bug 3113:[12] Processing by the WSDLPreprocessor produces output different depending on the JVM
  - Bug 3482:[13] wsa:From is not set correctly when service calls another service
  - Bug 3483:[14] xsd:anyType not serialized correctly
  - Bug 4432:[15] SimpleTopic.notify(SOAPElement element) drop child elements
  - Bug 4566:[16] globus-deploy-gar insists that ANT_HOME be set
  - Bug 4831:[17] GWSDL RP Inheritence Limited
  - Bug 5071:[18] Problems in building GT 4.0.4 with Java 1.6 and Ant 1.7

[7] http://bugzilla.globus.org/globus/show_bug.cgi?id=3513
[8] http://bugzilla.globus.org/globus/show_bug.cgi?id=2471
[9] http://bugzilla.globus.org/globus/show_bug.cgi?id=2445
[10] http://bugzilla.globus.org/globus/show_bug.cgi?id=2921
[11] http://bugzilla.globus.org/globus/show_bug.cgi?id=2926
[12] http://bugzilla.globus.org/globus/show_bug.cgi?id=3113
[13] http://bugzilla.globus.org/globus/show_bug.cgi?id=3482
[14] http://bugzilla.globus.org/globus/show_bug.cgi?id=3483
[15] http://bugzilla.globus.org/globus/show_bug.cgi?id=4432
[16] http://bugzilla.globus.org/globus/show_bug.cgi?id=4566
[17] http://bugzilla.globus.org/globus/show_bug.cgi?id=4831
[18] http://bugzilla.globus.org/globus/show_bug.cgi?id=5071

5. For More Information

Click here[19] for more information about this component.

[19] index.html

Chapter 7. GT 4.0 Java WS Core : System Administrator's Guide

1. Introduction

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

Important

This information is in addition to the basic Globus Toolkit prerequisite, overview, installation, security configuration instructions in the GT 4.0 System Administrator's Guide[1]. Read through this guide before continuing!

2. Building and Installing

Java WS Core is built and installed as part of a default GT 4.0 installation. For basic installation instructions, see the GT 4.0 System Administrator's Guide[2]. No extra installation steps are required for this component.

The following are optional instructions for more advanced types of installations. These are for those advanced users who want to build the latest code from CVS or are just interested in the Java WS Core.

2.1. Building from source

1. Obtain the source code for Java WS Core:

   From CVS.

   a. To get the latest source from cvs execute:

   ```
   cvs -d :pserver:anonymous@cvs.globus.org:/home/globdev/CVS/globus-packages \
       checkout wsrf
   ```

   b. Change into the `wsrf` directory.

   ```
   cd wsrf
   ```

   From Core-only source distribution.

   a. Untar or unzip the distribution archive.

   ```
   tar xvfz ws-core-XXX-src.tar.gz
   ```

   b. Change into the unpacked distribution directory.

[1] http://www.globus.org/toolkit/docs/4.0/admin/docbook/
[2] http://www.globus.org/toolkit/docs/4.0/admin/docbook/

```
cd ws-core-XXX
```

2. Set the `GLOBUS_LOCATION` environment variable to the absolute path of the target directory of your installation. On Windows:

   ```
   set GLOBUS_LOCATION=c:\gt4
   ```

   On Unix/Linux:

   ```
   setenv GLOBUS_LOCATION /soft/gt4/
   ```

   or

   ```
   export GLOBUS_LOCATION=/soft/gt4/
   ```

   If `GLOBUS_LOCATION` is not set, an `install` directory will be created under the current directory.

3. Run:

   ```
   ant all
   ```

   Additional arguments can be specified on the `ant` command line to customize the build:

   - `-DwindowsOnly=false` - generate launch scripts for standard Globus tools such as `grid-proxy-init`, etc. (Unix/Linux only)
   - `-Dall.scripts=true` - generate Windows and Unix launch scripts
   - `-Denable.container.desc=true` - create and configure the container with a global security descriptor

2.2. Installing Core-only binary distribution

1. Untar or unzip the distribution archive.

   ```
   tar xvfz ws-core-XXX-bin.tar.gz
   ```

2. Change into the unpacked distribution directory.

   ```
   cd ws-core-XXX
   ```

3. Set the `GLOBUS_LOCATION` environment variable to the unpacked distribution directory. On Windows:

   ```
   set GLOBUS_LOCATION=c:\gt4
   ```

   On Unix/Linux:

   ```
   setenv GLOBUS_LOCATION /soft/gt4/
   ```

   or

   ```
   export GLOBUS_LOCATION=/soft/gt4/
   ```

Note: Please make sure to have the JAAS[3] library installed if running with J2SE 1.3.1.

3. Configuring

3.1. Configuration overview

Java WS Core provides per- `gar` 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 support security might also have the *`security-config.xml`* (security deployment descriptor) file. Please see the Security Descriptor[4] page in the GT4 WS Authorization Framework documentation for details.

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

3.2. Syntax of the interface:

3.2.1. 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.

[3] http://java.sun.com/products/jaas/index-10.html
[4] http://www.globus.org/toolkit/docs/4.0/security/authzframe/security_descriptor.html

Table 7.1. General configuration parameters

Name	Value	Description	Comments
logicalHost	<hostname>	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.	Optional
disableDNS	<boolean>	This parameter specifies whether to perform DNS lookup on the `logicalHost` parameter. By default `"false"` is assumed (DNS lookup is performed).	Optional
domainName	<domain name>	This parameter specifies the domain name to append to the host name if the host name is not qualified by a domain.	Optional
publishHostName	<boolean>	This parameter specifies whether to publish the hostname or the ip address. It is only used when DNS lookups are enabled (`disableDNS` is false).	Optional
server.id	<string>	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 "`<ip>-<containerPort>`". In Tomcat, the server id will default to "`<ip>-<webApplicationName>`".	Optional (since GT 4.0.1)

Table 7.2. Standalone/embedded container-specific configuration parameters

Name	Value	Description	Comments
containerThreads	<int>	This parameter controls the initial thread pool size for the container. If not set, it defaults to 2. In GT 4.0.3+ it defaults to 1.	Optional
containerThreadsMax	<int>	This parameter sets the maximum number of threads for the container. By default it is set to 4 * the `containerThread` setting.	Optional
containerThreadsHighWaterMark	<int>	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.	Optional
containerTimeout	<int>	This parameter controls the container timeout. That is, the maximum amount of time the container will wait to receive a message from the client. By default it is set to 3 minutes.	Optional (since GT 4.0.2)
webContext	<name>	This parameter specifies the context name under which the services are published under: `http://<host>:<port>/<webContext>/services/MyService`. By default "`wsrf`" name is used. In Tomcat, this parameter is always set to the web application's name.	Optional (since GT 4.0.1)

Table 7.3. Default container thread pool settings

Type	Min. threads	Max. threads
standalone	2	8
embedded	2	8

Table 7.4. Default container thread pool settings (GT 4.0.3+ only)

Type	Min. threads	Max. threads
standalone	2	20
embedded	1	3

3.2.2. Service Configuration

3.2.2.1. WSDD

An example of a deployment descriptor for a CounterService:

```
<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 7.5. Axis Standard Parameters

Name	*Value*	*Description*	*Comments*
className	<class>	This parameter specifies a class that implements the web service methods.	Required
handlerClass	<class>	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.	Recommended in our environment
scope	<value>	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.	*Application* scope is recommended
wsdlFile	<path>	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.	Required in our environment
allowedMethods	<list of methods>	This parameter specifies a space or comma separated list of method names that can be called via SOAP. `"*"` indicates that all methods of the service class can be invoked via SOAP.	Optional. By default all methods are allowed.

Table 7.6. Java WS Core Parameters

Name	*Value*	*Description*	*Comments*
loadOnStartup	<boolean>	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 the Lifecycle and activation[5] section for details.	Optional
allowedMethodsClass	<class>	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.	Optional
providers	<list of providers>	This parameter specifies a space separated list of provider names or class names. Please see operation provider support[6] section for details. This parameter only has effect if used with *org.globus.axis.providers.RPCProvider* handlerClass.	Optional

Please see Custom Deployment[7] for details on Axis Web Services Deployment Descriptor.

3.2.2.2. JNDI

An example of a JNDI configuration bit for a CounterService:

```
<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>
```

[5] ../../common/javawscore/developer-index.html#Activation
[6] http://www.globus.org/toolkit/docs/4.0/common/javawscore/developer-index.html#s-javawscore-developer-OperationProvider
[7] http://ws.apache.org/axis/java/user-guide.html#PublishingServicesWithAxis

```
        </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 the JNDI section[8] 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.

3.2.2.2.1. ResourceHomeImpl

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

Table 7.7. ResourceHomeImpl parameters

Name	*Value*	*Description*	*Comments*
resourceKeyName	<qname>	This parameter specifies a QName of the resource key. The namespace is specified in the `{}`. For example, this QName will be used to discover the SOAP header that contains the key of the resource in the request.	Required
resourceKeyType	<class>	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.	Optional. Defaults to `java.lang.String`
resourceClass	<class>	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.	Required
sweeperDelay	<long>	This parameter specifies how often the resource sweeper runs in milliseconds.	Optional. Defaults to 1 minute
cacheLocation	<jndi path>	This parameter specifies the JNDI location of the resource cache for this resource home. Please see Configuring Resource Cache below for details.	Optional

3.2.2.2.1.1. Configuring Resource Cache

If *ResourceHomeImpl* is configured with resource class that implements the *PersistenceCallback* interface it will store the resource objects wrapped in Java SoftReference[9]. 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

[8] ../../common/javawscore/developer-index.html#s-javawscore-developer-JNDIDetails
[9] http://java.sun.com/j2se/1.4.2/docs/api/java/lang/ref/SoftReference.html

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:

```
<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>
  </resourceParams>
</resource>
```

In this case a LRU cache is configured. The "*timeout*" parameter (in ms) is used to specify the idle time of the resource object before it is removed from the cache. The same cache resource can be reused in different services but usually one cache per service will be configured.

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.

3.2.2.2.2. ServiceResourceHome

This implementation does not accept any special parameters.

3.2.3. Usage Statistics Configuration

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

The targets to which the usage statistics are sent to 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.

3.2.4. 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[10] with a -profile option to load configuration files in a specific profile.

4. Deploying

The Globus services can be run either in the standalone Java WS Core container that is installed with GT, or deployed into Tomcat.

4.1. Java WS Core container

The standalone Java WS Core container can be started and stopped with the provided **globus-start-container** and **globus-stop-container** programs. There are also helper programs (available only with the full GT installation) to start and stop the container detached from the controlling terminal (**globus-start-container-detached** and **globus-stop-container-detached**). These commands are documented in the Java WS Core Command Reference[11].

4.1.1. Deploying and undeploying services

To deploy a service into Java WS Core container use the **globus-deploy-gar** tool. To undeploy a service use **globus-undeploy-gar**. These commands are documented in the Java WS Core Command Reference[12].

[10] http://www.globus.org/toolkit/docs/4.0/common/javawscore/rn01re01.html
[11] http://www.globus.org/toolkit/docs/4.0/common/javawscore/Java_WS_Core_Commandline_Frag.html
[12] http://www.globus.org/toolkit/docs/4.0/common/javawscore/Java_WS_Core_Commandline_Frag.html

4.1.2. Recommended JVM settings for the Java WS Core container

It is recommended to increase the maximum heap size of the JVM when running the container. By default on Sun JVMs a 64MB maximum heap size is used. The maximum heap size can be set using the `-Xmx` JVM option. Example:

```
$ setenv GLOBUS_OPTIONS -Xmx512M
$ $GLOBUS_LOCATION/bin/globus-start-container
```

The above example will make the container start with maximum heap size set to 512MB.

It is also recommended to experiment with other JVM settings to improve performance. For example, the `-server` option on Sun JVMs enables a server VM which can deliver better performance for server applications.

4.2. Deploying into Tomcat

To deploy a Java WS Core installation into Tomcat run:

```
$ cd $GLOBUS_LOCATION
$ ant -f share/globus_wsrf_common/tomcat/tomcat.xml deploySecureTomcat -Dtomcat.dir=<tomca
```

Where <*tomcat.dir*> is an *absolute* path to the Tomcat installation directory. Also, `-Dwebapp.name=<name>` can be specified to set the name of the web application under which the installation will be deployed. By default "`wsrf`" web application name is used.

The `deploySecureTomcat` task will update an existing Tomcat deployment if Java WS Core was already deployed under the specified web application name. The `redeploySecureTomcat` task can be used instead to overwrite the existing deployment.

Note

Please note that during deployment a subset of the files from Java WS Core installation is copied into Tomcat. Also, the copied files in Tomcat might have different permissions then the originals.

In addition to the above deployment step you will also need to modify the Tomcat `<tomcat_root>/conf/server.xml` configuration file. In particular you will need to add the following configuration entries:

- Tomcat 4.1.x

  1. Add a HTTPS Connector in the <*Service name="Tomcat-Standalone"*> section and update the parameters appropriately with your local configuration:

     ```
     <Connector
       className="org.apache.catalina.connector.http.HttpConnector"
       port="8443" minProcessors="5" maxProcessors="75"
       authenticate="true" secure="true" scheme="https"
       enableLookups="true" acceptCount="10" debug="0">
         <Factory
            className="org.globus.tomcat.catalina.net.HTTPSServerSocketFactory"
             proxy="/path/to/proxy/file"
             cert="/path/to/certificate/file"
             key="/path/to/private/key/file"
             cacertdir="/path/to/ca/certificates/directory"/>
     </Connector>
     ```

In the above the `proxy`, `cert`, `key` and `cacertdir` attributes are optional. Furthermore, the `proxy` and the combination of `cert` and `key` attributes are mutually exclusive.

Important

The credentials and certificate configuration is used only by the connector and is not used by the rest of the web services stack in Globus Toolkit. To configure credentials for use in the toolkit, refer Security Descriptor[13].

2. Add a HTTPS Valve in the <*Engine name="Standalone"* ... > section:

```
<Valve className="org.globus.tomcat.catalina.valves.HTTPSValve"/>
```

- Tomcat 5.0.x

1. Add a HTTPS Connector in the <*Service name="Catalina"*> section and update the parameters appropriately with your local configuration:

```
<Connector
  className="org.globus.tomcat.coyote.net.HTTPSConnector"
  port="8443" maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
  autoFlush="true"
  disableUploadTimeout="true" scheme="https"
  enableLookups="true" acceptCount="10" debug="0"
   proxy="/path/to/proxy/file"
   cert="/path/to/certificate/file"
   key="/path/to/private/key/file"
   cacertdir="/path/to/ca/certificates/directory"/>
```

In the above the `proxy`, `cert`, `key` and `cacertdir` attributes are optional. Furthermore, the `proxy` and the combination of `cert` and `key` attributes are mutually exclusive.

Important

The credentials and certificate configuration is used only by the connector and is not used by the rest of the web services stack in Globus Toolkit. To configure credentials for use in the toolkit, refer Security Descriptor[14].

2. Add a HTTPS Valve in the <*Engine name="Catalina"* ... > section:

```
<Valve className="org.globus.tomcat.coyote.valves.HTTPSValve"/>
```

- Tomcat 5.5.x

1. Add a HTTPS Connector in the <*Service name="Catalina"*> section and update the parameters appropriately with your local configuration:

```
<Connector
  className="org.globus.tomcat.coyote.net.HTTPSConnector"
```

[13] http://www.globus.org/toolkit/docs/4.0/security/authzframe/security_descriptor.html
[14] http://www.globus.org/toolkit/docs/4.0/security/authzframe/security_descriptor.html

```
port="8443" maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
autoFlush="true"
disableUploadTimeout="true" scheme="https"
enableLookups="true" acceptCount="10" debug="0"
protocolHandlerClassName="org.apache.coyote.http11.Http11Protocol"
socketFactory="org.globus.tomcat.catalina.net.BaseHTTPSServerSocketFactory"
 proxy="/path/to/proxy/file"
 cert="/path/to/certificate/file"
 key="/path/to/private/key/file"
 cacertdir="/path/to/ca/certificates/directory"/>
```

In the above the `proxy`, `cert`, `key` and `cacertdir` attributes are optional. Furthermore, the `proxy` and the combination of `cert` and `key` attributes are mutually exclusive.

Important

The credentials and certificate configuration is used only by the connector and is not used by the rest of the web services stack in Globus Toolkit. To configure credentials for use in the toolkit, refer Security Descriptor[15].

2. Add a HTTPS Valve in the *<Engine name="Catalina" ... >* section:

```
<Valve className="org.globus.tomcat.coyote.valves.HTTPSValve55"/>
```

Also, you may have to edit `<tomcat.dir>/webapps/wsrf/WEB-INF/web.xml` if you are running Tomcat on a non-default port, i.e. not using port 8443 (HTTPS). For example, if you run Tomcat on port 443 using HTTPS then the WSRF servlet entry should be modified as follows:

```
<web-app>
...
    <servlet>
        <servlet-name>WSRFServlet</servlet-name>
        <display-name>WSRF Container Servlet</display-name>
        <servlet-class>
            org.globus.wsrf.container.AxisServlet
        </servlet-class>
        <init-param>
            <param-name>defaultProtocol</param-name>
            <param-value>https</param-value>
        </init-param>
        <init-param>
            <param-name>defaultPort</param-name>
            <param-value>443</param-value>
        </init-param>
        <load-on-startup>true</load-on-startup>
    </servlet>
...
</web-app>
```

[15] http://www.globus.org/toolkit/docs/4.0/security/authzframe/security_descriptor.html

Note

We recommend running Tomcat with Java 1.4.2+.

4.2.1. Enabling local invocations

To enable local invocation[16] in Tomcat you must add `axis-url.jar` to the CLASSPATH before starting Tomcat.

For example on Windows:

```
> cd <tomcat.dir>
> set CLASSPATH=<tomcat.dir>\common\lib\axis-url.jar
> bin\startup
```

On Unix/Linux (csh/tcsh):

```
$ cd <tomcat.dir>
$ setenv CLASSPATH <tomcat.dir>/common/lib/axis-url.jar
$ bin/startup
```

4.2.2. Debugging

4.2.2.1. Tomcat log files

Please always check the Tomcat log files under the `<tomcat.dir>/logs` directory for any errors or exceptions.

4.2.2.2. Enabling Log4J debugging

- Tomcat 4.1.x

  Copy `$GLOBUS_LOCATION/lib/commons-logging.jar` files to `<tomcat.dir>/common/lib` directory. Also, copy `<tomcat.dir>/webapps/wsrf/WEB-INF/classes/log4j.properties` file to `<tomcat.dir>/common/classes/` directory. Then configure the Log4j configuration file in `<tomcat.dir>/common/classes/` directory appropriately. The debugging settings will affect all the code in *all* web applications.

- Tomcat 5.0.x, 5.5.x

  Copy `$GLOBUS_LOCATION/lib/log4j-1.2.8.jar` and `$GLOBUS_LOCATION/lib/commons-logging.jar` files to `<tomcat.dir>/webapps/wsrf/WEB-INF/lib/` directory. Then configure the Log4j configuration file in `<tomcat.dir>/webapps/wsrf/WEB-INF/classes/` directory appropriately. The debugging settings will only affect the web application code.

4.2.3. Creating WAR file

To create a `.war` of a Java WS Core installation do:

```
$ cd $GLOBUS_LOCATION
$ ant -f share/globus_wsrf_common/tomcat/tomcat.xml war -Dwar.file=<war.file>
```

[16] http://www.globus.org/toolkit/docs/4.0/common/javawscore/developer-index.html#s-javawscore-developer-LocalInvocations

Where <*war.file*> specifies the *absolute* path of the war file.

Please note that deploying a `war` file might not be enough to have a working Java WS Core deployment. For example, in some cases the `xalan.jar` must be placed in the `endorsed` directory of the container.

4.2.4. Deploying and undeploying services

To deploy a service into Tomcat, first deploy the service into a regular GT installation using the **globus-deploy-gar** tool and then deploy the GT installation into Tomcat (as described in Section 4, "Deploying"). Similarly, to undeploy a service, first undeploy the service from a regular GT installation using **globus-undeploy-gar** tool and then *deploy* the GT installation into Tomcat.

Note

Some GT services may not work properly in Tomcat.

5. Testing

To execute Java WS Core tests first ensure Ant is configured with JUnit (To install JUnit with Ant copy the junit.jar found in the JUnit distribution to the $ANT_HOME/lib directory).

To execute the test do the following:

1. Start the standalone container with `-nosec` argument:

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

2. Run the interoperability tests:

```
$ ant -f share/globus_wsrf_test/runtests.xml runServer \
      -Dtests.jar=$GLOBUS_LOCATION/lib/wsrf_test_interop.jar
```

3. Run the unit tests:

```
$ ant -f share/globus_wsrf_test/runtests.xml runServer \
      -Dtests.jar=$GLOBUS_LOCATION/lib/wsrf_test_unit.jar -DbasicTestsOnly=true
```

4. If some tests failed examine the test results in the `$GLOBUS_LOCATION/share/globus_wsrf_test/tests/test-reports/` directory.

Please see the developer guide[17] for more information on running the tests and the testing infrastructure.

[17] http://www.globus.org/toolkit/docs/4.0/common/javawscore/developer-index.html#s-javawscore-developer-runningtests

6. Security Considerations

6.1. 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.

6.2. 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.

6.3. 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 Java WS Core Configuration for details).

7. Troubleshooting

7.1. `globus-stop-container` fails with an authorization error

By default `globus-stop-container` must be executed with the same credentials as the container it is running with. If the *ShutdownService* or the container is configured with separate private key and certificate files (usually `/etc/grid-security/containercert.pem` and `/etc/grid-security/containerkey.pem`) do the following to stop the container:

```
$ grid-proxy-init -cert /etc/grid-security/containercert.pem \
                  -key /etc/grid-security/containerkey.pem \
                  -out containerproxy.pem
$ setenv X509_USER_PROXY containerproxy.pem
$ globus-stop-container
$ unsetenv X509_USER_PROXY
$ rm containerproxy.pem
```

Alternatively, the *ShutdownService* can be configured with a separate gridmap file to allow a set of users to stop the container. Please see the WS Authentication & Authorization[18] section for details.

[18] ../../security/wsaa.html

7.2. `globus-start-container` hangs during startup

By default Sun 1.4.x+ JVMs are configured to use `/dev/random` device as an entropy source. Sometimes the machine can run out of entropy and applications (such as the container) using the `/dev/random` device will block until more entropy is available. One workaround for this issue is to configure the JVM to use `/dev/urandom` (non-blocking) device instead. For Sun JVMs a `java.security.egd` system property can be set to configure a different entropy source. To set the system property and pass it to `globus-start-container` script do the following:

```
export GLOBUS_OPTIONS=-Djava.security.egd=file:/dev/urandom
```

or

```
setenv GLOBUS_OPTIONS -Djava.security.egd=file:/dev/urandom
```

The same issue can also affect client programs. If you are running a client program with a GT generated script, you can set the `GLOBUS_OPTIONS` environment property as described above. However, if you are using a custom script or directly launching a program using the `java` command line, make sure to set the `java.security.egd` system property explicitly on the `java` command line. For example:

```
java -classpath $CLASSPATH -Djava.security.egd=file:/dev/urandom my.package.FooProgram
```

Note: This does not apply to Windows machines.

7.3. Programs fail with `java.lang.NoClassDefFoundError: javax/security/...` errors

These errors might occur when running with J2SE 1.3.1 and the JAAS[19] library is not installed. Either install the JAAS[20] library or upgrade to J2SE 1.4.x or higher.

7.4. `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)
```

[19] http://java.sun.com/products/jaas/index-10.html
[20] http://java.sun.com/products/jaas/install_notes.html

```
at org.apache.coyote.tomcat5.CoyoteRequestFacade.setAttribute(CoyoteRequestFacade.java:35
at org.globus.tomcat.coyote.valves.HTTPSValve.expose(HTTPSValve.java:99)
```

These exceptions will prevent the transport security to work properly in Tomcat.

This is a Tomcat bug. Keep restarting Tomcat until it starts without the `ConcurrentModificationException` or switch to a different version of Tomcat.

7.5.java.net.SocketException: Invalid argument or cannot assign requested address

If you see the "`java.net.SocketException: Invalid argument or cannot assign requested address`" error in the container log or on the client side try setting the following property:

```
$ export GLOBUS_OPTIONS="-Djava.net.preferIPv4Stack=true"
```

7.6. General troubleshooting information

In general, if you want to investigate a problem on your own please see the Debugging and Logging[21] section for details on how to turn on debugging. Also, please note that most of the command line clients have a *`-debug`* option that will display more detailed error messages, including the error stack traces. Also, searching the mailing lists[22] such as discuss@globus.org[23] or developer-discuss@globus.org[24] (before posting a message) can also be very fruitful. Finally, if you think you have found a bug please report it in our Bugzilla[25] system. Please include as much as detail about the problem as possible.

8. Usage statistics collection by the Globus Alliance

The following usage statistics are sent by Java WS Core by default in a UDP packet (in addition to the Java WS Core component code, packet version, timestamp, and the source IP address):

- On container startup:
  - container id - random number
  - container type - standalone, servlet, or unknown
  - event type - container startup
  - list of services - service names only
- On container shutdown:
  - container id - random number
  - container type - standalone, servlet, or unknown
  - event type - container shutdown

[21] http://www.globus.org/toolkit/docs/4.0/common/javawscore/developer-index.html#s-javawscore-developer-debugging
[22] http://www.globus.org/email-archive-search.php
[23] mailto:discuss@globus.org
[24] mailto:developer-discuss@globus.org
[25] http://bugzilla.globus.org/bugzilla/

- list of activated services - service names only (4.0.3+)
- container uptime (4.0.3+)

If you wish to disable this feature, please see the Java WS Core System Administrator's Guide section on Usage Statistics Configuration for instructions.

Also, please see our policy statement[26] on the collection of usage statistics.

[26] http://www.globus.org/toolkit/docs/4.0/Usage_Stats.html

Chapter 8. GT 4.0 Java WS Core : User's Guide

1. Introduction

The Java WS Core User's Guide provides general end user-oriented information.

2. Command-line tools

Please see the Java WS Core Command Reference.

3. Graphical user interfaces

There is no support for this type of interface for Java WS Core.

4. Troubleshooting

4.1. Running clients from any directory

A client launched directly through the `java` executable might fail if ran from a directory other then the `GLOBUS_LOCATION` (It usually happens if the client receives notifications). To ensure that a client can be started from any directory pass a `GLOBUS_LOCATION` system property on the `java` command line set to the appropriate `GLOBUS_LOCATION` directory. Also, make sure to put the `GLOBUS_LOCATION` directory as the very first entry in the classpath used for the application.

For example on Unix/Linux:

```
$ java -DGLOBUS_LOCATION=$GLOBUS_LOCATION -classpath $GLOBUS_LOCATION:mylib.jar foo.MyCla
```

or on Windows:

```
> java -DGLOBUS_LOCATION=%GLOBUS_LOCATION% -classpath %GLOBUS_LOCATION%;mylib.jar foo.MyC
```

4.2. Program fails with "Failed to acquire notification consumer home instance from registry" error

Please see Section 4.1, “Running clients from any directory” if a client fails with "`Failed to acquire notification consumer home instance from registry. Caused by javax.naming.NameNotFoundException: Name services is not bound in this Context`" error.

4.3. Container warning: "The WS-Addressing 'To' request header is missing"

This warning is logged by the container if the request did not contain the necessary *WS-Addressing* headers. The client either did not attempt to send those headers at all or is somehow misconfigured (for example, the client used an incorrect configuration file). If you using a Java client and launching it directly using the `java` executable take a look at Section 4.1, “Running clients from any directory”.

4.4. java.io.IOException: Token length X > 33554432

If you see the "`java.io.IOException: Token length X > Y`" error in the container log it usually means you are trying to connect to HTTPS server using HTTP. For example, the service address specifies `8443` as a port number and `http` as the protocol name. In general, `8443` port number should be used with the `https` protocol, and `8080` port number should be used with the `http` protocol.

4.5. The standalone container appears to hang

If the standalone container appears to hang, for example the list of deployed services is not shown after a while or all requests to the container time out, please see the Debugging hanged Java process section for information on how to diagnose this problem.

4.6. org.globus.wsrf.InvalidResourceKeyException: Argument key is null/Resource key is missing

The "`org.globus.wsrf.InvalidResourceKeyException: Argument key is null`" (or "`org.globus.wsrf.InvalidResourceKeyException: Resource key is missing`") error usually indicates 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).

Make sure that the EPR used to invoke the service contains the appropriate resource key. If you are using some command-line tool make sure to specify the resource key using the `-k` option or pass a complete EPR from a file using the `-e` option.

4.7. General troubleshooting information

In general, if you want to investigate a problem on your own please see the Debugging and Logging[1] section for details on how to turn on debugging. Also, please note that most of the command line clients have a *`-debug`* option that will display more detailed error messages, including the error stack traces. Also, searching the mailing lists[2] such as discuss@globus.org[3] or developer-discuss@globus.org[4] (before posting a message) can also be very fruitful. Finally, if you think you have found a bug please report it in our Bugzilla[5] system. Please include as much as detail about the problem as possible.

[1] http://www.globus.org/toolkit/docs/4.0/common/javawscore/developer-index.html#s-javawscore-developer-debugging
[2] http://www.globus.org/email-archive-search.php
[3] mailto:discuss@globus.org
[4] mailto:developer-discuss@globus.org
[5] http://bugzilla.globus.org/bugzilla/

5. Miscellaneous information

5.1. JVM entropy source

In GT 4.0.0 to 4.0.3 on Unix/Linux machines the shell scripts generated for the various Java command line tools specified the entropy generator for the JVM in an incorrect way. That caused the JVM to fallback to a slower method of obtaining the entropy and resulted in slower startup of these programs.

The fix for this issue is very simple and can be applied to any existing GT 4.0.x installation. To apply the fix first download update.xml[6] Ant script, and ensure your `GLOBUS_LOCATION` environment property is properly set. Than execute (from any directory):

```
ant -f update.xml
```

After the fix is applied the startup time of the command line clients can improve by up to 60%. For more information please see Bug 4721[7].

5.2. Running Java Programs From Command Line

Sometimes it might be necessary to run a Java program directly using the `java` executable. There are two recommended ways of doing so (the `GLOBUS_LOCATION` environment variable must first be set in both cases):

Important

If you are not using any of these two methods please take a look at Section 4.1, “Running clients from any directory”.

5.2.1. With `globus-devel-env` script help

The `globus-devel-env` script can be used to set the proper `CLASSPATH` environment variable. To set the `CLASSPATH` on Windows execute:

```
> %GLOBUS_LOCATION%\etc\globus-devel-env.bat
```

On Unix/Linux machines execute (for bash/sh):

```
$ . $GLOBUS_LOCATION/etc/globus-devel-env.sh
```

or (for csh/tcsh):

```
$ source $GLOBUS_LOCATION/etc/globus-devel-env.csh
```

Once the `globus-devel-env` is executed successfully run the Java program, for example:

On Windows:

```
> java -DGLOBUS_LOCATION=%GLOBUS_LOCATION% foo.MyClass
```

On Unix/Linux:

```
$ java -DGLOBUS_LOCATION=$GLOBUS_LOCATION foo.MyClass
```

[6] http://bugzilla.globus.org/bugzilla/attachment.cgi?id=1063&action=view
[7] http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=4721

Note: Passing `-DGLOBUS_LOCATION` is not necessary but will enable the client to execute from any directory.

5.2.2. Using bootstrap

Sometimes the above method might fail if the `CLASSPATH` environment variable is too long for the OS to handle. With the bootstrap method, a bootstrap code is executed first which sets the classpath programmatically and then invokes the specified Java program. To invoke a Java program on Windows through bootstrap execute:

```
> java -cp %GLOBUS_LOCATION%\lib\bootstrap.jar -DGLOBUS_LOCATION=%GLOBUS_LOCATION% \
      org.globus.bootstrap.Bootstrap foo.MyClass
```

On Unix/Linux machines execute:

```
$ java -cp $GLOBUS_LOCATION/lib/bootstrap.jar -DGLOBUS_LOCATION=$GLOBUS_LOCATION \
      org.globus.bootstrap.Bootstrap foo.MyClass
```

6. Usage statistics collection by the Globus Alliance

The following usage statistics are sent by Java WS Core by default in a UDP packet (in addition to the Java WS Core component code, packet version, timestamp, and the source IP address):

- On container startup:
  - container id - random number
  - container type - standalone, servlet, or unknown
  - event type - container startup
  - list of services - service names only
- On container shutdown:
  - container id - random number
  - container type - standalone, servlet, or unknown
  - event type - container shutdown
  - list of activated services - service names only (4.0.3+)
  - container uptime (4.0.3+)

If you wish to disable this feature, please see the Java WS Core System Administrator's Guide section on Usage Statistics Configuration for instructions.

Also, please see our policy statement[8] on the collection of usage statistics.

[8] http://www.globus.org/toolkit/docs/4.0/Usage_Stats.html

Chapter 9. GT 4.0 Java WS Core : Developer's Guide

1. 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.

2. Before you begin

2.1. Feature summary

New Features in the GT 4.0 release

- Implementation of the 2004/06 OASIS *WSRF* and *WSN* working draft specifications (with minor fixes to the 1.2-draft-01 published schemas and with the March 2004 version of the *WS-Addressing* specification)
- Basic HTTP/1.1 client & server support
- *JNDI*-based registry based on the JNDI service in Apache Tomcat
- An implementation of the Work Manager[1] and Timer[2] specifications

Other Supported Features

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

Deprecated Features

- None

2.2. Tested platforms

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

Tested platforms for Java WS Core:

- Linux (Red Hat 7.3)

[1] http://dev2dev.bea.com/wlplatform/commonj/twm.csp
[2] http://dev2dev.bea.com/wlplatform/commonj/twm.csp

- Windows 2000 and XP
- Solaris 9

Tested JVMs for Java WS Core:

- Sun JVM[3] 1.3.1, 1.4.2 and 1.5.0
- IBM JVM[4] 1.3.1, 1.4.1, and 1.4.2
- BEA JRockit JVM[5] 1.5.0

JVM notes:

- GCJ[6] is not supported.
- If using IBM JVM 1.4.1 please see bug 2828[7] for more information.

Tested containers for Java WS Core:

- Java WS Core container
- Tomcat 4.1.31
- Tomcat 5.0.30

2.3. Backward compatibility summary

Protocol changes since GT version 3.2

- HTTP/1.1 with 'chunked' transfer encoding is now used by default.
- Wire messages follow the new schemas and therefore are completely different (see below).

API changes since GT version 3.2

- The majority of the APIs are new. Some APIs resemble GT 3.2 APIs, for example `ServiceData` is replaced by `ResourceProperty` and `ServiceDataSet` is replaced by `ResourcePropertySet`.

Schema changes since GT version 3.2

- Schemas are completely new. The WS Java Core implements the OASIS WSRF and WSN working drafts specifications (with minor fixes to the 1.2-draft-01 published schemas and with the March 2004 version of the WS-Addressing specification.)

2.4. Technology dependencies

Java WS Core depends on the following GT components:

- Java CoG Kit[8]

[3] http://java.sun.com/j2se/
[4] http://www-106.ibm.com/developerworks/java/jdk/
[5] http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/jrockit
[6] http://gcc.gnu.org/java/
[7] http://bugzilla.globus.org/globus/show_bug.cgi?id=2828#c4
[8] http://www.globus.org/cog/java/

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

- Apache Xerces[9]
- Apache XML Security[10]
- Apache Axis[11]
- Apache Xalan[12]
- Apache Addressing[13]
- Apache Commons BeanUtils[14]
- Apache Commons CLI[15]
- Apache Commons Collections[16]
- Apache Commons Digester[17]
- Concurrent Library[18]
- Apache Tomcat JNDI[19]

Please see the Technology Dependencies Details page[20] for details.

2.5. Security Considerations

2.5.1. 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.

2.5.2. 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.

[9] http://xml.apache.org/xerces2-j/
[10] http://xml.apache.org/security/
[11] http://ws.apache.org/axis/
[12] http://xml.apache.org/xalan-j/
[13] http://ws.apache.org/ws-fx/addressing/
[14] http://jakarta.apache.org/commons/beanutils/
[15] http://jakarta.apache.org/commons/cli/
[16] http://jakarta.apache.org/commons/collections/
[17] http://jakarta.apache.org/commons/digester/
[18] http://gee.cs.oswego.edu/dl/classes/EDU/oswego/cs/dl/util/concurrent/intro.html
[19] http://cvs.apache.org/viewcvs.cgi/jakarta-tomcat-catalina/catalina/src/share/org/apache/naming/
[20] dependencies.html

2.5.3. 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 Java WS Core Configuration for details).

3. Architecture and design overview

- Java WS Core Design Document [doc[21] | pdf[22]]
- Java WS Core UML [vsd[23] | Resource (.gif)[24], Resource Property (gif)[25], Notification (gif)[26]]
- Java WS Core Notification UML Sequence Diagrams [vsd[27] | Subscription (jpg)[28], Notification (jpg)[29]]

4. Public interface

The semantics and syntax of the APIs and WSDL for the component, along with descriptions of domain-specific structured interface data, can be found in the Java WS Core Public Interfaces Chapter.

5. Usage scenarios

5.1. Basics

5.1.1. WSDL/SOAP rules

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.

5.1.1.1. 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).

[21] developer/JavaWSCoreDesign.doc
[22] developer/JavaWSCoreDesign.pdf
[23] developer/JavaWSCoreUML.vsd
[24] developer/Resource.gif
[25] developer/ResourceProperty.gif
[26] developer/Notification.gif
[27] developer/NotificationSequences.vsd
[28] developer/NotificationSequence1.jpg
[29] developer/NotificationSequence2.jpg

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.

5.1.1.2. SOAP Encoding

Do *not* use or mix the literal mode with the SOAP encoding mode (R2706[30]). For example, do not use the `soapenc:Array` type. Please see the 5.2.3 section[31] in the *WS-I Basic Profile* for details.

5.1.2. Operation providers and its configuration

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 `handleClass` 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. If provider names are used, they must first be defined as parameters in the `<globalConfiguration>` element of the *main* deployment descriptor (`etc\globus_wsrf_core\server-config.wsdd`).

   For example:

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

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:

[30] http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html#refinement16638080
[31] http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html#refinement16556272

```
<service name="SubscriptionManagerService"  provider="Handler" use="literal" style="docume
  <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.

5.1.3. JNDI configuration file

Java WS Core provides Tomcat's JNDI[32] implementation. The file format of Java WS Core's `jndi-config.xml` is slightly different from the Tomcat's `server.xml` file. One main difference is that the `<resourceParams>` are specified as children of `<resource>` objects. Also, Java WS Core's `jndi-config.xml` parser is case sensitive and all element names are lowercase.

All elements defined in the `<global>` section of the JNDI configuration file are deployed into the `java:comp/env` context under the name specified in the 'name' attribute. All `<service>` elements are deployed into the `java:comp/env/services/<service name>` context. New objects and contexts can be added or modified dynamically at runtime but they will not be persisted. The only way to always have an object around is to deploy it in the `jndi-config.xml` file. All services *share* the same `java:comp/env` context. This is different from EJBs where each EJB has a separate `java:comp/env` context.

When deploying `<resource>` in `jndi-config.xml`, make sure to use the `org.globus.wsrf.tools.jndi.BeanFactory` as a BeanFactory (value of a 'factory' parameter) instead of `org.apache.naming.factory.BeanFactory`. Your bean must have a default constructor. If your bean implements the `org.globus.wsrf.jndi.Initializable` interface, the `initialize()` function will be automatically called after all parameters are set on the bean.

Please see The JNDI Tutorial[33] for more information on JNDI programming.

5.1.4. Lifecycle and activation

5.1.4.1. Activating a service

To activate a service, an RPCProvider is available from both Axis and Globus.

5.1.4.2. Activating a service using the Axis RPCProvider

The `scope` setting of the service dictates when and how service instances are created:

[32] http://jakarta.apache.org/tomcat/tomcat-5.0-doc/jndi-resources-howto.html
[33] http://java.sun.com/products/jndi/tutorial/

Table 9.1. Scope settings

Application	One instance of the service is used for all invocations.
Request	One instance is created per invocation. This is the default (if scope parameter is not set in the deployment descriptor).
Session	One instance is created per session.

If the service implements the javax.xml.rpc.server.ServiceLifecycle[34] 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.

With Axis RPCProvider, JAAS credentials are never associated with the invocation thread.

5.1.4.3. Activating a service using the Globus RPCProvider

The `scope` setting of the service dictates when and how service instances are created (only *Application* and *Request* scopes are supported with Globus RPCProvider):

Table 9.2. Scope settings and activation

Application	Service/provider instances are created either on first invocation or on container startup. The behavior is determined by the value of the "loadOnStartup" 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 Section 5.2.2.3, “Obtaining standard MessageContext properties” for details). If service or providers implement the javax.xml.rpc.server.ServiceLifecycle[35] interface, the lifecycle methods will be called accordingly.
Request	One instance is created per invocation. This is the default (if scope parameter is not set in the deployment descriptor). Behaves more or less just like the Axis RPCProvider (service/providers instances are created per invocation, ServiceLifecycle methods called right before and after service method invocation, no JAAS credentials during ServiceLifecycle methods).

5.1.4.4. Activating a ResourceHome

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.wsrf.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.wsrf.jndi.BeanFactory* class for its factory and is defined directly under a *service* entry in a *jndi-config.xml* file.

[34] http://java.sun.com/j2ee/1.4/docs/api/javax/xml/rpc/server/ServiceLifecycle.html
[35] http://java.sun.com/j2ee/1.4/docs/api/javax/xml/rpc/server/ServiceLifecycle.html

5.1.4.5. Activating ServiceResourceHome

If you are using a ServiceResourceHome please make sure to deploy the service with the *"loadOnStartup"* option enabled and in *Application* scope. That will ensure that the ResourceHome is initialized with the right service/resource.

5.2. Programming

5.2.1. General

5.2.1.1. Using Apache Addressing API

The WS-RF and WS-N specifications distributed with Java WS Core use *WS-Addressing* (the March 2004 version of the specification) for addressing services and resources. Java WS Core uses the Apache Addressing[36] library for WS-Addressing support. The API is pretty straightforward and easy to use. Most of the work is done in `Addressing-Handler` deployed in the client and server configuration files. See Apache Addressing documentation[37] for details.

5.2.1.1.1. Call object

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:

```
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[]);
```

5.2.1.1.2. AddressingLocator class

The Apache Addressing library also contains a version of Axis' WSDL2Java tool. It extends the Axis' WSDL2Java tool functionality by generating, in addition to all the regular classes, the `<service>Addressing` interface and `<service>AddressingLocator` class.

The `AddressingLocator` class can be used to get a stub for a service by passing the Apache Addressing `EndpointReferenceType` parameter.

For example:

```
String url = "http://localhost:8080/axis/services/Version";
```

[36] http://ws.apache.org/ws-fx/addressing/
[37] http://ws.apache.org/ws-fx/addressing/

```
EndpointReferenceType epr = new EndpointReferenceType();
epr.setAddress(new Address(url));

VersionServiceAddressingLocator locator =
     new VersionServiceAddressingLocator();

VerionServicePortType port = locator.getVersionPort(epr);

port.getVersion();
```

5.2.1.1.3. ReferenceProperties

In the WS-RF and WS-N specifications, the WS-Addressing `ReferenceProperties` are used to carry resource identity information. The resource identity can be anything as long as it serializes as a XML element. The `ReferenceProperties` 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 property.

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

```
// create a reference property
QName keyName = new QName("http://axis.org", "VersionKey");
String keyValue = "123";

SimpleResourceKey key = new SimpleResourceKey(keyName, keyValue);

ReferencePropertiesType props = new ReferencePropertiesType();

// convert to SOAPElement and add to the list
props.add(key.toSOAPElement());
...
```

Then pass it to `AddressingHeaders`:

```
...
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.setReferenceProperties(props);

// 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[]);
```

Or set it on `EndpointReferenceType`:

```
...
String url = "http://localhost:8080/axis/services/Version";

EndpointReferenceType epr = new EndpointReferenceType();
epr.setAddress(new Address(url));
epr.setProperties(props);

VersionServiceAddressingLocator locator =
    new VersionServiceAddressingLocator();

VerionServicePortType port = locator.getVersionPort(epr);

port.getVersion();
```

5.2.1.2. Setting up and receiving notifications (Notification Consumer)

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

5.2.1.2.1. Step 1: Implement the callback

The notification consumer application must provide an implementation of the `org.globus.wsrf.NotifyCallback` interface. The `deliver` function of the interface will be invoked whenever a notification for that consumer arrives. The `message` parameter will either be of `org.w3c.dom.Element` type if the message type was unknown or some object mapped to the type of the message.

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).

5.2.1.2.2. 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 Section 4.2, “Program fails with "`Failed to acquire notification`

consumer home instance from registry" error" 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.

5.2.1.2.3. 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 {
  ....
}
```

5.2.1.2.4. Step 4: Subscribe to the callback

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

Example:

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

TopicExpressionType topicExpression = new TopicExpressionType();
topicExpression.setDialect(WSNConstants.SIMPLE_TOPIC_DIALECT);
topicExpression.setValue(WSRFConstants.TERMINATION_TIME);

Subscribe request = new Subscribe();
request.setUseNotify(Boolean.TRUE);
request.setConsumerReference(consumerEPR);
request.setTopicExpression(topicExpression);

SubscribeResponse subResponse = port.subscribe(request);
...
```

5.2.1.2.5. 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:

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

SubscriptionManagerServiceAddressingLocator sLocator = new SubscriptionManagerServiceAddre
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:

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

5.2.2. Service-side specific

5.2.2.1. Obtaining container and service endpoint information

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 `ServiceHost` 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 `ResourceContext` API. For example:

```
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 `AddressingUtils` API. For example:

```
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.

5.2.2.2. Obtaining service parameters

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. Java WS Core provides some helper functions to ease this process:

```
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.

Note

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

5.2.2.3. Obtaining 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.
- *org.apache.axis.Constants.MC_REMOTE_ADDR* - the IP address of the client.
- *org.apache.axis.MessageContext.TRANS_URL* - the URL of the request.

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.

5.2.2.4. Making local calls

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 and only when a Axis MessageContext is associated with a current thread. URLs with "*local*" protocol name are used for local invocations.

To invoke a service locally, do the following:

1. Register "*local*" protocol handler:

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

2. Create a service URL with "*local*" protocol:

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

3. Configure the service stub for local invocation and make the call:

```
MyServiceAddressingLocator locator =
       new MyServiceAddressingLocator();
MyService port = locator.getMyServicePort(url);

LocalTransportUtils.enableLocalTransport((Stub)port);

port.hello();
```

5.2.3. Client-side specific

5.2.3.1. Client notes

Any program that is based on Java WS Core should contain as a first entry in its classpath the directory of the Java WS Core installation. This is to ensure that the right *client-config.wsdd* is used by the client. That configuration file contains important client-side information such as handlers, type mappings, etc.

Also, any program that is a notification consumer should be initialized with the appropriate `GLOBUS_LOCATION` system property (set to the installation directory of Java WS Core). If the system property is not set, the notification consumer might not initialize or work properly.

5.3. Deploying

5.3.1. Grid Archive (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. The GAR files are deployed using globus-deploy-gar (globus-deploy-gar(1)) and undeployed using globus-undeploy-gar (globus-undeploy-gar(1)) tools.

5.3.1.1. GAR file structure

Table 9.3. GAR file structure

`docs/`	This directory contains service documentation files.
`share/`	This directory contains files that can be accessed or used by all services.
`schema/`	This directory contains service WSDL and schema files.
`etc/`	This directory contains service configuration files and a `post-deploy.xml` Ant script.
`bin/`	This directory contains service executables such as command line tools, GUI, etc.
`lib/`	This directory contains service and third party library files and any LICENSE files.
`server-deploy.wsdd`	This file is the server side deployment descriptor.
`client-deploy.wsdd`	This file is the client side deployment descriptor.
`jndi-config-deploy.xml`	This file is the JNDI configuration file.

5.3.1.2. Deployment process

The contents of the GAR file are processed in the following way (all steps are performed only if necessary):

- Any files in the `docs/` directory in the GAR are copied into the `$GLOBUS_LOCATION/docs/<gar.id>/` directory.
- Any files in the `share/` directory in the GAR are copied into the `$GLOBUS_LOCATION/share/<gar.id>/` directory.
- Any files in the `schema/` directory in the GAR are copied into the `$GLOBUS_LOCATION/share/schema/` directory.

- Any files in the `etc/` directory in the GAR are copied into the `$GLOBUS_LOCATION/etc/<gar.id>/` directory.
- Any files in the `bin/` directory in the GAR are copied into the `$GLOBUS_LOCATION/bin/` directory.
- Any `.jar` files in the `lib/` directory of the GAR are copied into the `$GLOBUS_LOCATION/lib/` directory.
- Any file that contains the word "LICENSE" in the name in the `lib/` directory of the GAR is copied into the `$GLOBUS_LOCATION/share/licenses/` directory.
- The `server-deploy.wsdd` in the GAR is copied to `$GLOBUS_LOCATION/etc/<gar.id>/server-config.wsdd`. If a profile name was specified during deployment, the `server-deploy.wsdd` will be copied to `$GLOBUS_LOCATION/etc/<gar.id>/<profile.name>-server-config.wsdd`. The `server-config.wsdd` file will be set with user-only access permissions.
- The `jndi-config-deploy.xml` in the GAR is copied to `$GLOBUS_LOCATION/etc/<gar.id>/jndi-config.xml`. If a profile name was specified during deployment the `jndi-config-deploy.xml` will be copied to `$GLOBUS_LOCATION/etc/<gar.id>/<profile.name>-jndi-config.xml`. The `jndi-config.xml` file will be set with user only-access permissions.
- The `client-deploy.wsdd` in the GAR is merged into a common `$GLOBUS_LOCATION/client-config.wsdd` file.
- An undeploy script (`$GLOBUS_LOCATION/etc/globus_packages/<gar.id>/undeploy.xml`) is created.
- A `etc/post-deploy.xml` Ant script is called if the GAR contains one. The *setup* target is called automatically.

Notes:

- If the `post-deploy.xml` script creates some files, they will *not* be removed by `undeploy`.
- During deployment, filtering is done for contents of the `server-deploy.wsdd` and `jndi-config-deploy.xml` files to replace the `@config.dir@` token with the "`etc/<gar.id>`" value, and the `@gar.id@` token with the "`<gar.id>`" value.

5.3.1.3. Creating a GAR file through Ant

5.3.1.3.1. Creating GAR file

To create a GAR file use the following example:

```
<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 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.

5.3.1.3.2. Deploying GAR file

To deploy a GAR file use the following example:

```
<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.

5.3.1.3.3. Undeploying GAR file

To undeploy a GAR file use the following example:

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

The `gar.id` property must be passed. This property specifies the base name of the gar to undeploy.

5.3.2. Generating launcher scripts

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:

```
...
<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:

```
<target name="generateUnixScripts" if="generate.unix" depends="testUnix">
  <ant antfile="${build.launcher}" target="generateLauncher">
    ...
    <property name="default.jvm.options"
              value="-DFOO=&quot;$FOO&quot;${DELIM}-DBAR=&quot;$BAR&quot;/>
  </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(1)).

5.4. Testing

Tests in the Java WS Core are based on the JUnit[38] 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`.

5.4.1. Writing Tests

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.

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

[38] http://www.junit.org/

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.

5.4.2. Running Tests

To execute the tests on the Java WS Core install, run the following (assuming the tests have been deployed into that install):

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

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

By default, the tests that use a container will try to access a container running at `http://localhost:8080/wsrf/services/`.

To specify a different container, use the `-Dtest.server.url=<url>` property.

To execute PackageTests only, specify `-DbasicTestsOnly=true`.

To execute SecurityTests only, specify `-DsecurityTestsOnly=true`.

The test reports will be put in the `$GLOBUS_LOCATION/share/globus_wsrf_test/tests/test-reports` directory by default. A different test reports directory can be specified by passing `-Djunit.reports.dir=<directory>`.

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

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

5.5. Other

5.5.1. Adding 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:

5.5.1.1. Step 1: Implement the evaluator

Table 9.4. Evaluator interfaces

If the evaluator is a...	then it must implement:
query expression evaluator	`org.globus.wsrf.query.ExpressionEvaluator`
topic expression evaluator	`org.globus.wsrf.topicexpression.TopicExpressionEvaluator`

5.5.1.2. 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.

5.5.1.2.1. 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.wsrf.jndi.BeanFactory</value>
      </parameter>
    </resourceParams>
  </resource>
</global>
```

Where the <resource> attribute:

name	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.
type	Specifies the class that implements the expression evaluator.

5.5.1.2.2. Registering topic expression evaluators

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

Example:

```
<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:

name	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.
type	Specifies the class that implements the expression evaluator.

5.5.1.3. 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.

5.5.1.3.1. 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:

```
<typeMapping
   encodingStyle=""
   deserializer="org.apache.axis.encoding.ser.SimpleDeserializerFactory"
   serializer="org.apache.axis.encoding.ser.SimpleSerializerFactory"
   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.

5.5.1.3.2. 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:

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

Note

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

5.5.1.4. Step 4: Configuring a helper serializer for GetCurrentMessageProvider

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:

```
<global>
  <resource
    name="providers/GetCurrentMessageProvider/foo.bar.MyNotificationMessage"
    type="foo.bar.MyMessageSerializer">
    <resourceParams>
```

```
      <parameter>
        <name>factory</name>
        <value>org.globus.wsrf.jndi.BeanFactory</value>
      </parameter>
    </resourceParams>
  </resource>
</global>
```

Where the `<resource>` attribute:

`name`	Must start with *`providers/GetCurrentMessageProvider/`* and must end with the full class name of the notification message.
`type`	Specifies the class that implements the `org.globus.wsrf.encoding.ObjectConverter` interface and is responsible for serializing the notification message. The `GetCurrentMessageProvider` will use the type of the notification message to find the helper serializer.

6. Tutorials

- The Globus Toolkit 4 Programmer's Tutorial by Borja Sotomayor[39]
- Basic Grid Development Tools Tutorial[40]
- GT3.2 to GT4 Migration: A First HOWTO by Terry Harmer and Julie McCabe[41]

7. Debugging

7.1. Logging

Logging in the Java WS Core is based on the Jakarta Commons Logging[42] API. Commons Logging provides a consistent interface for instrumenting source code while at the same time allowing the user to plug-in a different logging implementation. Currently we use Log4j[43] as a logging implementation. Log4j uses a separate configuration file to configure itself. Please see Log4j documentation for details on the configuration file format[44].

Java WS Core is deployed with two Log4j configuration files:

- `$GLOBUS_LOCATION/container-log4j.properties` (configures logging for the standalone container)
- `$GLOBUS_LOCATION/log4j.properties` (configures logging for everything else besides the standalone container)

[39] http://gdp.globus.org/gt4-tutorial/
[40] http://ds.informatik.uni-marburg.de/MAGE/gdt/tutorial.html
[41] http://www.qub.ac.uk/escience/howtos/GT3%20to%20GT4%20Version%200.3.htm
[42] http://jakarta.apache.org/commons/logging/
[43] http://logging.apache.org/log4j/
[44] http://logging.apache.org/log4j/docs/api/org/apache/log4j/PropertyConfigurator.html#doConfigure(java.lang.String, org.apache.log4j.spi.LoggerRepository)

7.2. Tracing SOAP messages

7.2.1. 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:

```
<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:

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

7.2.2. 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:

```
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.

7.2.3. 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:

```
$ 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.

7.3. 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[45]
- IBM JDK 5.0: Java Diagnostics Guide[46]

[45] http://java.sun.com/j2se/1.5/pdf/jdk50_ts_guide.pdf
[46] http://publib.boulder.ibm.com/infocenter/javasdk/v5r0

7.4. 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.

7.5. 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.

8. Troubleshooting

8.1. No socket factory for 'https' protocol

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)
```

add the following to the client:

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

8.2. 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. Ensure that the GT4 installation directory is listed as a first entry in the `CLASSPATH` of the client. For example:

```
CLASSPATH=/usr/local/globus-4.0.0:/foo/bar/others.jar:...
```

If you are seeing this problem in Tomcat, copy the `client-config.wsdd` from the GT4 installation directory to the Web application's `WEB-INF/classes` directory.

8.3. java.lang.ClassNotFoundException: org/apache/xml/security/transforms/implementations/FuncHere

The above error indicates that the wrong version of Xalan is being used. This happens with Some JDK versions and can be fixed by placing the correct version in the endorsed directory.

$GLOBUS_LOCATION/endorsed has the correct jars. You can either put the jar in the endorsed directory of the JVM or specify -Djava.endorsed.dir property to point to the endorsed directory. Clients that are shipped as a part of the toolkit should have the property specified by default.

8.4. General troubleshooting information

In general, if you want to investigate a problem on your own please see the Debugging and Logging[47] section for details on how to turn on debugging. Also, please note that most of the command line clients have a *`-debug`* option that will display more detailed error messages, including the error stack traces. Also, searching the mailing lists[48] such as discuss@globus.org[49] or developer-discuss@globus.org[50] (before posting a message) can also be very fruitful. Finally, if you think you have found a bug please report it in our Bugzilla[51] system. Please include as much as detail about the problem as possible.

9. Related Documentation

Overview material about WSRF and WSN and more information on the implemented WSDL and schema can be found here[52].

Information about ongoing standards work can be found here:

- WS-Addressing[53]
- WS-ResourceFramework[54]
- WS-Notification[55]

[47] http://www.globus.org/toolkit/docs/4.0/common/javawscore/developer-index.html#s-javawscore-developer-debugging
[48] http://www.globus.org/email-archive-search.php
[49] mailto:discuss@globus.org
[50] mailto:developer-discuss@globus.org
[51] http://bugzilla.globus.org/bugzilla/
[52] http://www.globus.org/wsrf/
[53] http://www.w3.org/2002/ws/addr/
[54] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf
[55] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn

10. Appendices

10.1. 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:

```
$ 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.

Chapter 10. GT 4.0 Component Fact Sheet: Java Web Services Core (Java WS Core)

1. Brief component overview

The Java WS Core is an implementation of the *Web Services Resource Framework (WSRF)* and the *Web Service Notification (WSN)* family of standards. It provides APIs and tools for building stateful Web services.

2. Summary of features

New Features in the GT 4.0 release

- Implementation of the 2004/06 OASIS *WSRF* and *WSN* working draft specifications (with minor fixes to the 1.2-draft-01 published schemas and with the March 2004 version of the *WS-Addressing* specification)
- Basic HTTP/1.1 client & server support
- *JNDI*-based registry based on the JNDI service in Apache Tomcat
- An implementation of the Work Manager[1] and Timer[2] specifications

Other Supported Features

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

Deprecated Features

- None

3. Usability summary

Usability improvements for Java WS Core:

- A number of performance and scalability improvements:
    - Optimized basic messaging performance

[1] http://dev2dev.bea.com/wlplatform/commonj/twm.csp
[2] http://dev2dev.bea.com/wlplatform/commonj/twm.csp

  - Optimized querying of resource property document
  - Scalability improvements of subscriptions
  - And more
- Per-service configuration and configuration profiles
- Convenient command line scripts around deployment and undeployment Ant scripts

4. Backward compatibility summary

Protocol changes since GT version 3.2

- HTTP/1.1 with 'chunked' transfer encoding is now used by default.
- Wire messages follow the new schemas and therefore are completely different (see below).

API changes since GT version 3.2

- The majority of the APIs are new. Some APIs resemble GT 3.2 APIs, for example `ServiceData` is replaced by `ResourceProperty` and `ServiceDataSet` is replaced by `ResourcePropertySet`.

Schema changes since GT version 3.2

- Schemas are completely new. The WS Java Core implements the OASIS WSRF and WSN working drafts specifications (with minor fixes to the 1.2-draft-01 published schemas and with the March 2004 version of the WS-Addressing specification.)

5. Technology dependencies

Java WS Core depends on the following GT components:

- Java CoG Kit[3]

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

- Apache Xerces[4]
- Apache XML Security[5]
- Apache Axis[6]
- Apache Xalan[7]
- Apache Addressing[8]
- Apache Commons BeanUtils[9]

[3] http://www.globus.org/cog/java/
[4] http://xml.apache.org/xerces2-j/
[5] http://xml.apache.org/security/
[6] http://ws.apache.org/axis/
[7] http://xml.apache.org/xalan-j/
[8] http://ws.apache.org/ws-fx/addressing/
[9] http://jakarta.apache.org/commons/beanutils/

- Apache Commons CLI[10]
- Apache Commons Collections[11]
- Apache Commons Digester[12]
- Concurrent Library[13]
- Apache Tomcat JNDI[14]

Please see the Technology Dependencies Details page[15] for details.

6. Tested platforms

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

Tested platforms for Java WS Core:

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

Tested JVMs for Java WS Core:

- Sun JVM[16] 1.3.1, 1.4.2 and 1.5.0
- IBM JVM[17] 1.3.1, 1.4.1, and 1.4.2
- BEA JRockit JVM[18] 1.5.0

JVM notes:

- GCJ[19] is not supported.
- If using IBM JVM 1.4.1 please see bug 2828[20] for more information.

Tested containers for Java WS Core:

- Java WS Core container
- Tomcat 4.1.31
- Tomcat 5.0.30

[10] http://jakarta.apache.org/commons/cli/
[11] http://jakarta.apache.org/commons/collections/
[12] http://jakarta.apache.org/commons/digester/
[13] http://gee.cs.oswego.edu/dl/classes/EDU/oswego/cs/dl/util/concurrent/intro.html
[14] http://cvs.apache.org/viewcvs.cgi/jakarta-tomcat-catalina/catalina/src/share/org/apache/naming/
[15] dependencies.html
[16] http://java.sun.com/j2se/
[17] http://www-106.ibm.com/developerworks/java/jdk/
[18] http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/jrockit
[19] http://gcc.gnu.org/java/
[20] http://bugzilla.globus.org/globus/show_bug.cgi?id=2828#c4

7. Associated standards

Associated standards for Java WS Core:

- WS-ResourceProperties[21]
- WS-ResourceLifetime[22]
- WS-ServiceGroup[23]
- WS-BaseFaults[24]
- WS-BaseNotification[25]
- WS-Topics[26]
- WS-Addressing[27]
- WS-I Basic Profile[28]

8. For More Information

Please see Java WS Core documentation[29] for more information.

[21] http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf
[22] http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-03.pdf
[23] http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ServiceGroup-1.2-draft-02.pdf
[24] http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-BaseFaults-1.2-draft-02.pdf
[25] http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-03.pdf
[26] http://docs.oasis-open.org/wsn/2004/06/wsn-WS-Topics-1.2-draft-01.pdf
[27] http://msdn.microsoft.com/ws/2004/03/ws-addressing
[28] http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html
[29] index.html

Chapter 11. GT 4.0 Component Guide to Public Interfaces: Java WS Core

1. Semantics and syntax of APIs

1.1. 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.

1.2. 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].

2. Semantics and syntax of the WSDL

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

- WSRF[2]
  - WS-ResourceProperties (WSRF-RP) specification[3] [wsdl[4] | xsd[5]]
  - WS-ResourceLifetime (WSRF-RL) specification[6] [wsdl[7] | xsd[8]]
  - WS-ServiceGroup (WSRF-SG) specification[9] [wsdl[10] | xsd[11]]
  - WS-BaseFaults (WSRF-BF) specification[12] [wsdl[13] | xsd[14]]
- WSN[15]
  - WS-BaseNotification specification[16] [wsdl[17] | xsd[18]]
  - WS-Topics specification[19] [xsd[20]]

[1] http://www.mcs.anl.gov/~gawor/javawscore/globus_4_0_branch/doc/javadocs/
[2] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf
[3] http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf
[4] http://viewcvs.globus.org/viewcvs.cgi/wsrf/schema/wsrf/properties/WS-ResourceProperties.wsdl?rev=1.14&only_with_tag=globus_4_0_0
[5] http://viewcvs.globus.org/viewcvs.cgi/wsrf/schema/wsrf/properties/WS-ResourceProperties.xsd?rev=1.7&only_with_tag=globus_4_0_0
[6] http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-03.pdf
[7] http://viewcvs.globus.org/viewcvs.cgi/wsrf/schema/wsrf/lifetime/WS-ResourceLifetime.wsdl?rev=1.11&only_with_tag=globus_4_0_0
[8] http://viewcvs.globus.org/viewcvs.cgi/wsrf/schema/wsrf/lifetime/WS-ResourceLifetime.xsd?rev=1.6&only_with_tag=globus_4_0_0
[9] http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ServiceGroup-1.2-draft-02.pdf
[10] http://viewcvs.globus.org/viewcvs.cgi/wsrf/schema/wsrf/servicegroup/WS-ServiceGroup.wsdl?rev=1.9&only_with_tag=globus_4_0_0
[11] http://viewcvs.globus.org/viewcvs.cgi/wsrf/schema/wsrf/servicegroup/WS-ServiceGroup.xsd?rev=1.6&only_with_tag=globus_4_0_0
[12] http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-BaseFaults-1.2-draft-02.pdf
[13] http://viewcvs.globus.org/viewcvs.cgi/wsrf/schema/wsrf/faults/WS-BaseFaults.wsdl?rev=1.5&only_with_tag=globus_4_0_0
[14] http://viewcvs.globus.org/viewcvs.cgi/wsrf/schema/wsrf/faults/WS-BaseFaults.xsd?rev=1.8&only_with_tag=globus_4_0_0
[15] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn
[16] http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-03.pdf
[17] http://viewcvs.globus.org/viewcvs.cgi/wsrf/schema/wsrf/notification/WS-BaseN.wsdl?rev=1.11&only_with_tag=globus_4_0_0
[18] http://viewcvs.globus.org/viewcvs.cgi/wsrf/schema/wsrf/notification/WS-BaseN.xsd?rev=1.6&only_with_tag=globus_4_0_0
[19] http://docs.oasis-open.org/wsn/2004/06/wsn-WS-Topics-1.2-draft-01.pdf
[20] http://viewcvs.globus.org/viewcvs.cgi/wsrf/schema/wsrf/notification/WS-Topics.xsd?rev=1.4&only_with_tag=globus_4_0_0

2.1. Resource Properties

- WS-ResourceProperties (WSRF-RP) specification[21] [wsdl[22] | xsd[23]]

3. Command-line tools

Please see the Java WS Core Command Reference.

4. Overview of Graphical User Interface

There is no support for this type of interface for Java WS Core.

5. Semantics and syntax of domain-specific interface

There is no content available at this time.

6. Configuration interface

Please see Java WS Core Configuration.

7. Environment variable interface

Table 11.1. Globus standard environment variables

Name	*Value*	*Description*	*Comments*
GLOBUS_LOCATION	<path>	The <path> is the root location of the Java WS Core installation. Must be an absolute path.	Required
GLOBUS_TCP_PORT_RANGE	<min,max>	The <min,max> 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.	Optional
GLOBUS_UDP_SOURCE_PORT_RANGE	<min,max>	The <min,max> is the minimum and maximum port range for UDP outgoing sockets (useful for systems behind firewalls).	Optional (since GT 4.0.1)
GLOBUS_HOSTNAME	<host>	The <host> is either a hostname or ip address. The host ip address under which the container and services will be exposed.	Optional

[21] http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf
[22] http://viewcvs.globus.org/viewcvs.cgi/wsrf/schema/wsrf/properties/WS-ResourceProperties.wsdl?rev=1.14&only_with_tag=globus_4_0_0
[23] http://viewcvs.globus.org/viewcvs.cgi/wsrf/schema/wsrf/properties/WS-ResourceProperties.xsd?rev=1.7&only_with_tag=globus_4_0_0

Table 11.2. Launch script specific environment variables

Name	*Value*	*Description*	*Comments*
GLOBUS_OPTIONS	<arguments>	The <arguments> are arbitrary arguments that can be passed to the JVM. See below for a detailed list of supported options.	Optional
JAVA_HOME	<path>	The <path> 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.	Optional
CLASSPATH	<classpath>	This environment property is ignored by launch scripts.	Ignored

Table 11.3. Options supported by the `GLOBUS_OPTIONS` environment property

Name	*Value*	*Description*	*Comments*
-Dorg.globus.wsrf.proxy.port	int	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.	
-Dorg.globus.wsrf.container.server.id	string	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. In GT 4.0.0, by default the container will store the persistent resources under the `~/.globus/persisted/<ip>/` directory. Therefore, if there are multiple containers running as the same user on the same machine the same directory will be used for the persistent resources. That might cause the containers to overwrite each other's persistent data. However, this conflict is less likely in GT 4.0.1+ because of better server id defaults. In GT 4.0.1+, by default the standalone container will store the persistent resources under the `~/.globus/persisted/<ip>-<containerPort>` directory. While in Tomcat the `~/.globus/persisted/<ip>-<webApplicationName>` 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/<server.id>/` instead.	
-Dorg.globus.wsrf.container.persistence.dir	directory	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.	Since GT 4.0.2

Chapter 12. GT 4.0 Java WS Core: Quality Profile

1. Test coverage reports

- Clover report[1]

2. Code analysis reports

- PMD report[2]
- FindBugs report[3]

3. Outstanding bugs

- Bug 2471:[4] Message security signature verification issues
- Bug 2445:[5] Same input and output messages in WSDL confuse Axis
- Bug 2921:[6] Support for TerminationTimeChangeRejectedFault
- Bug 2926:[7] Local transport does not work without a current MessageContext
- Bug 3113:[8] Processing by the WSDLPreprocessor produces output different depending on the JVM
- Bug 3482:[9] wsa:From is not set correctly when service calls another service
- Bug 3483:[10] xsd:anyType not serialized correctly

4. Bug Fixes

- Bug 2650[11]
- Bug 2651[12]
- Bug 2664[13]

[1] http://www.mcs.anl.gov/~gawor/javawscore/HEAD/clover/clover-reports-html/
[2] http://www.mcs.anl.gov/~gawor/javawscore/HEAD/pmd/pmd_report.html
[3] http://www.mcs.anl.gov/~gawor/javawscore/HEAD/findbugs/findbugs_report.html
[4] http://bugzilla.globus.org/globus/show_bug.cgi?id=2471
[5] http://bugzilla.globus.org/globus/show_bug.cgi?id=2445
[6] http://bugzilla.globus.org/globus/show_bug.cgi?id=2921
[7] http://bugzilla.globus.org/globus/show_bug.cgi?id=2926
[8] http://bugzilla.globus.org/globus/show_bug.cgi?id=3113
[9] http://bugzilla.globus.org/globus/show_bug.cgi?id=3482
[10] http://bugzilla.globus.org/globus/show_bug.cgi?id=3483
[11] http://bugzilla.globus.org/globus/show_bug.cgi?id=2650
[12] http://bugzilla.globus.org/globus/show_bug.cgi?id=2651
[13] http://bugzilla.globus.org/globus/show_bug.cgi?id=2664

- Bug 2765[14]
- Bug 2814[15]
- Bug 2828[16]
- Bug 2854[17]
- Bug 2885[18]
- Bug 2887[19]
- Bug 2923[20]
- Bug 2924[21]
- Bug 3080[22]
- Bug 3162[23]
- Bug 3200[24]

5. Performance reports

- Basic messaging performance[25]
- WSRF/WSN operation performance[26]

[14] http://bugzilla.globus.org/globus/show_bug.cgi?id=2765
[15] http://bugzilla.globus.org/globus/show_bug.cgi?id=2814
[16] http://bugzilla.globus.org/globus/show_bug.cgi?id=2828
[17] http://bugzilla.globus.org/globus/show_bug.cgi?id=2854
[18] http://bugzilla.globus.org/globus/show_bug.cgi?id=2885
[19] http://bugzilla.globus.org/globus/show_bug.cgi?id=2887
[20] http://bugzilla.globus.org/globus/show_bug.cgi?id=2923
[21] http://bugzilla.globus.org/globus/show_bug.cgi?id=2924
[22] http://bugzilla.globus.org/globus/show_bug.cgi?id=3080
[23] http://bugzilla.globus.org/globus/show_bug.cgi?id=3162
[24] http://bugzilla.globus.org/globus/show_bug.cgi?id=3200
[25] coreMessagingPerformance.gif
[26] java_ws_core_wsrf_perf.html

Chapter 13. GT 4.0 Samples for Java WS Core

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 notification 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 -z none
   ```

   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 > e
```

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:allowedMethodsC
 <ns1:scope>Application</ns1:scope>
 <ns1:wsdlFile>share/schema/core/registry/registry_service.wsdl</ns1:wsdlFile>
 <ns1:provider>Handler</ns1:provider>
</ns1:ServiceWSDD>
```

The above command queries the `ManagementService` and returns the WSDD information of the `ContainerRegistrySer`

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

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

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

```
<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:

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

Then run the following command:

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

The above command updates the `providers` resource property of the `ContainerRegistryService` resource of `Management`
That results in the `QueryRPProvider` provider to be removed from the operation provider list of the `ContainerRegistrySe`

6. Now perform the query operation again:

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

Since the operation provider for the `QueryResourceProperties` operation was removed the above command should generate th
error message:

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

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

Chapter 14. GT 4.0 Migrating Guide for Java WS Core

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

1. Migrating from GT2

Not applicable. This component did not exist in GT2.

2. Migrating from GT3

2.1. Key Migration Points

2.1.1. Schemas

2.1.1.1. GWSDL

GT3 used `GWSDL` to describe service messages and operations. GT4 uses standard WSDL[1] with one extensibility attribute to describe resource properties. Please see the WS-ResourceProperties[2] specification for details on expressing the resource properties in `WSDL`.

2.1.1.2. Port Types

In GT3 every service inherited some basic operations and functionality (from `GridService` port type) as required by the OGSI specification. However, in GT4 there is no such requirement (because the WSRF[3]/WSN[4] specifications do not require it). Also, the `Factory` port type defined in the OGSI specification does not exist in WSRF/WSN specifications. Therefore, there is no standard create operation or functionality provided by GT4.

2.1.1.3. WSDL formatting

GT3 relied on *wrapped* formatting of the WSDL. GT4 uses standard *document* formatting. This change introduces differences in how the Java interface for the service looks. Please see the design document[5] and the document/literal[6] section for more details.

2.1.2. Developing services

In GT3, the business logic of the service and the state were coupled together in one class. In GT4, the business logic and the state are decoupled and placed in two separate classes. The business logic is put in a *service* class while the state is put in a *resource* class. The *service* is stateless while the *resource* is stateful. Also, GT4 introduces ResourceHome[7] classes that are responsible for managing and discovering resources. Please see the programming model overview[8] for details.

[1] http://www.globus.org/toolkit/docs/4.0/common/javawscore/Java_WS_Core_Glossary.html#wsdl
[2] http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf
[3] http://www.globus.org/toolkit/docs/4.0/common/javawscore/Java_WS_Core_Glossary.html#wsrf
[4] http://www.globus.org/toolkit/docs/4.0/common/javawscore/Java_WS_Core_Glossary.html#wsn
[5] http://www.globus.org/toolkit/docs/4.0/common/javawscore/developer-index.html#s-javawscore-developer-archdes
[6] http://www.globus.org/toolkit/docs/4.0/common/javawscore/developer-index.html#s-javawscore-developer-DocumentLiteral
[7] http://www.globus.org/toolkit/docs/4.0/common/javawscore/Java_WS_Core_Glossary.html#ResourceHome
[8] http://www.globus.org/toolkit/docs/4.0/common/javawscore/Java_WS_Core_Public_Interfaces.html#s-javawscore-Public_Interfaces-apis

Even though a GT4 developer needs to modify two or three separate files the coding effort is about the same as in GT3.

2.1.3. Configuring services

In GT3, most of the configuration information was stored in the `server-config.wsdd` file. In GT4, since the business logic and state were decoupled, the service information is still kept in server-config.wsdd[9] but resource information is now placed in the new jndi-config.xml[10] file. Please see the JNDI[11] section for more details.

2.2. Related Documentation

- GT3.2 to GT4 Migration: A First HOWTO by Terry Harmer and Julie McCabe[12]

[9] http://www.globus.org/toolkit/docs/4.0/common/javawscore/Java_WS_Core_Glossary.html#server-config.wsdd
[10] http://www.globus.org/toolkit/docs/4.0/common/javawscore/Java_WS_Core_Glossary.html#jndi-config.xml
[11] http://www.globus.org/toolkit/docs/4.0/common/javawscore/developer-index.html#s-javawscore-developer-JNDIDetails
[12] http://www.qub.ac.uk/escience/howtos/GT3%20to%20GT4%20Version%200.3.htm

Chapter 15. Technology Dependencies Details For Java WS Core

1. Dependencies Details

Distributed w/ Library	Filename	Version	URL
Java CoG Kit FX	cog-axis.jar		
Java CoG Kit FX	cog-tomcat.jar		
Java CoG Kit	cog-jglobus.jar	1.2	http://www.globus.org/cog/java/
Java CoG Kit	cog-url.jar	1.2	http://www.globus.org/cog/java/
Java CoG Kit	jgss.jar	1.2	http://www.globus.org/cog/java/
Java CoG Kit	log4j-1.2.8.jar	1.2.8	http://jakarta.apache.org/log4j/
Java CoG Kit	jce-jdk13-125.jar	1.25	http://www.bouncycastle.org/
Java CoG Kit	**puretls.jar** [1]	0.9b4	http://www.rtfm.com/puretls/
Java CoG Kit	**cryptix32.jar**[1]	3.2.0	http://www.cryptix.org/
Java CoG Kit	**cryptix-asn1.jar**[1]		http://www.rtfm.com/puretls/
Java CoG Kit	cryptix.jar		http://www.cryptix.org/
Apache Xerces 2.6.2	xercesImpl.jar	2.6.2	http://xml.apache.org/xerces2-j/
Apache Xerces 2.6.2	xml-apis.jar	2.6.2	http://xml.apache.org/xerces2-j/
Apache Xerces 2.6.2	resolver.jar	2.6.2	http://xml.apache.org/xerces2-j/
Apache XML Security 1.2.1	xmlsec.jar	1.2.1	http://xml.apache.org/security/
Apache Axis 1.2	**axis.jar**[1]	1.2	http://ws.apache.org/axis/
Apache Axis 1.2	jaxrpc.jar		http://ws.apache.org/axis/
Apache Axis 1.2	saaj.jar		http://ws.apache.org/axis/
Apache Axis 1.2	commons-discovery.jar		http://jakarta.apache.org/commons/discovery/
Apache Axis 1.2	commons-logging.jar		http://jakarta.apache.org/commons/logging/
Apache Axis 1.2	wsdl4j.jar		http://sourceforge.net/projects/wsdl4j/
Apache Xalan	xalan.jar	2.6.0	http://xml.apache.org/xalan-j/
Apache WS-Addressing	**addressing-1.0.jar** [1]		http://ws.apache.org/addressing/
Apache WS-Security	**wss4j.jar** [2]		http://ws.apache.org/wss4j/
Apache Directory	naming-core-0.8.jar	0.8	http://directory.apache.org/subprojects/naming/
Apache Directory	naming-factory-0.8.jar	0.8	http://directory.apache.org/subprojects/naming/
Apache Directory	naming-java-0.8.jar	0.8	http://directory.apache.org/subprojects/naming/

Distributed w/ Library	Filename	Version	URL
Apache Directory	naming-resources-0.8.jar	0.8	http://directory.apache.org/subprojects/naming/
Apache Tomcat Servlet API	servlet.jar		http://jakarta.apache.org/tomcat/
Apache Commons	commons-beanutils.jar	1.6.1	http://jakarta.apache.org/commons/beanutils/
Apache Commons	**commons-cli-2.0.jar** [1]	~2.0	http://jakarta.apache.org/commons/cli/
Apache Commons	commons-collections-3.0.jar	3.0	http://jakarta.apache.org/commons/collections/
Apache Commons	commons-digester.jar	1.5	http://jakarta.apache.org/commons/digester/
util.concurrent library	concurrent.jar	1.3.4	http://gee.cs.oswego.edu/dl/classes/EDU/oswego/cs/dl/util/concurrent/intro.html
Timer and Work Manager API	commonj.jar	1.1	http://dev2dev.bea.com/wlplatform/commonj/twm.html
OpenSAML	**opensaml.jar** [1]		http://www.opensaml.org/

2. Legend

1. Custom-modified version
2. Custom version

3. Source code details

File(s)	Repository	Module	Tag	Source tarball	Misc.
cog-jglobus.jar, cog-url.jar	CoG CVS	jglobus	globus_4_0_branch		
jgss.jar	CoG CVS	jgss	HEAD		
cog-axis.jar, cog-tomcat.jar	CoG FX CVS	src/jglobus-fx/gsi	HEAD		
puretls.jar	CoG CVS	puretls-0.9b4	HEAD		
cryptix32.jar	CoG CVS	cryptix32	HEAD		
cryptix-asn1.jar	CoG CVS	cryptix-asn1	HEAD		
axis.jar	Apache SVN	/webservices/axis/tags/gt395		ws-axis.tar.gz[1]	with patch[2]
addressing-1.0.jar	Apache SVN	/webservices/addressing/tags/gt395		ws-addressing.tar.gz[3]	with patch[4]

[2] http://fisheye.globus.org/viewrep/~raw,r=globus_4_0_branch/GlobusToolkit/wsrf/java/lib-src/ws-axis/patch
[1] http://fisheye.globus.org/viewrep/~raw,r=globus_4_0_branch/GlobusToolkit/wsrf/java/lib-src/ws-axis/ws-axis.tar.gz
[4] http://fisheye.globus.org/viewrep/~raw,r=globus_4_0_branch/GlobusToolkit/wsrf/java/lib-src/ws-addressing/patch
[3] http://fisheye.globus.org/viewrep/~raw,r=globus_4_0_branch/GlobusToolkit/wsrf/java/lib-src/ws-addressing/ws-addressing.tar.gz

File(s)	Repository	Module	Tag	Source tarball	Misc.
wss4j.jar	Apache SVN	/webser-vices/wss4j/tags/gt395		ws-wss4j.tar.gz[5]	
commons-cli-2.0.jar	Apache SVN	/jakarta/commons/prop-er/cli/trunk	412903	commons-cli.tar.gz[6]	
opensaml.jar	GT CVS	opensaml	HEAD		

4. Repositories

Repository Name	Link
CoG CVS Root	cvs.globus.org:/homes/dsl/cog/CVS[7]
CoG FX CVS Root	cvs.cogkit.org:/cvs/cogkit[8]
GT CVS Root	cvs.globus.org:/home/globdev/CVS/globus-packages[9]
Apache SVN Root	http://svn.apache.org/repos/asf/[10]

[5] http://viewcvs.globus.org/viewcvs.cgi/*checkout*/wsrf/java/lib-src/ws-wss4j/ws-wss4j.tar.gz?&content-type=application/x-tar
[6] http://fisheye.globus.org/viewrep/~raw,r=MAIN/GlobusToolkit/wsrf/java/lib-src/commons-cli/commons-cli.tar.gz
[7] http://viewcvs.globus.org/viewcvs.cgi/?cvsroot=Java+COG
[8] http://www.cogkit.org/viewcvs/viewcvs.cgi/
[9] http://viewcvs.globus.org/viewcvs.cgi/?cvsroot=Globus+Packages
[10] http://svn.apache.org/viewcvs.cgi/

GT 4.0: Java WS Core Command Reference

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 15. Options

-help	Displays help information about the command.
-p <port>	Sets the port number for the container.
-quiet	Does not show a list of services at startup.
-debug	Enables debug mode.
-nosec	Starts a non secure (HTTP) container. Please note that this option only disables transport security. Message security still can be used.
-containerDesc <file>	Specifies a container security descriptor file.
-profile <name>	Specifies a configuration profile name for the container.

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 16. Common options

-h, --help	Displays help information about the command.
-d, --debug	Enables debug mode. For example, full stack traces of errors will be displayed.
-e, --eprFile <file>	Specifies an *XML* file that contains the *WS-Addressing* endpoint reference.
-s, --service <url>	Specifies the service URL.
-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.
-z, --authorization <type>	Specifies authorization type. **type** can be '**self**', '**host**', '**none**', or a string specifying the expected identity of the remote party.

Table 17. Shutdown options

'**soft**'	This option lets the threads die naturally.
'**hard**'	This option forces an immediate JVM shutdown.

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.

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.

Name

wsrf-destroy -- Destroys a resource

```
wsrf-destroy
```

Tool description

Destroys a resource.

Command syntax

```
wsrf-destroy [options]
```

Table 18. Common options

-h, --help	Displays help information about the command.
-d, --debug	Enables debug mode. For example, full stack traces of errors will be displayed.
-e, --eprFile <file>	Specifies an *XML* file that contains the *WS-Addressing* endpoint reference.
-s, --service <url>	Specifies the service URL.
-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.
-z, --authorization <type>	Specifies authorization type. **type** can be **'self'**, **'host'**, **'none'**, or a string specifying the expected identity of the remote party.

Example:

```
$ wsrf-destroy -s http://localhost:8080/wsrf/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'
```

Table 19. Common options

-h, --help	Displays help information about the command.
-d, --debug	Enables debug mode. For example, full stack traces of errors will be displayed.
-e, --eprFile <file>	Specifies an *XML* file that contains the *WS-Addressing* endpoint reference.
-s, --service <url>	Specifies the service URL.
-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.
-z, --authorization <type>	Specifies authorization type. **type** can be **'self'**, **'host'**, **'none'**, or a string specifying the expected identity of the remote party.

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

Table 20. Command-specific options

-u, --utc	Display time in UTC.

Example:

```
$ wsrf-set-termination-time -s http://localhost:8080/wsrf/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 21. Common options

-h, --help	Displays help information about the command.
-d, --debug	Enables debug mode. For example, full stack traces of errors will be displayed.
-e, --eprFile <file>	Specifies an *XML* file that contains the *WS-Addressing* endpoint reference.
-s, --service <url>	Specifies the service URL.
-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.
-z, --authorization <type>	Specifies authorization type. **type** can be '**self**', '**host**', '**none**', or a string specifying the expected identity of the remote party.

Examples:

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


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


$ wsrf-query -s http://localhost:8080/wsrf/services/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 22. Common options

-h, --help	Displays help information about the command.
-d, --debug	Enables debug mode. For example, full stack traces of errors will be displayed.
-e, --eprFile <file>	Specifies an *XML* file that contains the *WS-Addressing* endpoint reference.
-s, --service <url>	Specifies the service URL.
-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.
-z, --authorization <type>	Specifies authorization type. **type** can be '**self**', '**host**', '**none**', or a string specifying the expected identity of the remote party.

Example:

```
$ wsrf-get-property -s http://localhost:8080/wsrf/services/CounterService \
   -k "{http://counter.com}CounterKey" 123 \
  "{http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-01.xsd}Curr
```

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 23. Common options

-h, --help	Displays help information about the command.
-d, --debug	Enables debug mode. For example, full stack traces of errors will be displayed.
-e, --eprFile <file>	Specifies an *XML* file that contains the *WS-Addressing* endpoint reference.
-s, --service <url>	Specifies the service URL.
-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.
-z, --authorization <type>	Specifies authorization type. **type** can be '**self**', '**host**', '**none**', or a string specifying the expected identity of the remote party.

Example:

```
$ wsrf-get-properties -s http://localhost:8080/wsrf/services/CounterService \
  -k "{http://counter.com}CounterKey" 123 \
  "{http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-01.xsd}Curr
  "{http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-01.xsd}Term
```

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 24. Common options

-h, --help	Displays help information about the command.
-d, --debug	Enables debug mode. For example, full stack traces of errors will be displayed.
-e, --eprFile <file>	Specifies an *XML* file that contains the *WS-Addressing* endpoint reference.
-s, --service <url>	Specifies the service URL.
-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.
-z, --authorization <type>	Specifies authorization type. **type** can be '**self**', '**host**', '**none**', or a string specifying the expected identity of the remote party.

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>


$ wsrf-insert-property -s http://localhost:8080/wsrf/services/WidgetService \
   -k "{http://www.globus.org/namespaces/2004/06/core}WidgetKey" 123 \
   in.xml
```

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 25. Common options

-h, --help	Displays help information about the command.
-d, --debug	Enables debug mode. For example, full stack traces of errors will be displayed.
-e, --eprFile <file>	Specifies an *XML* file that contains the *WS-Addressing* endpoint reference.
-s, --service <url>	Specifies the service URL.
-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.
-z, --authorization <type>	Specifies authorization type. **type** can be '**self**', '**host**', '**none**', or a string specifying the expected identity of the remote party.

Example: Contents of **in.xml**:

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


$ wsrf-update-property -s http://localhost:8080/wsrf/services/WidgetService \
   -k "{http://www.globus.org/namespaces/2004/06/core}WidgetKey" 123 \
   in.xml
```

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 26. Common options

-h, --help	Displays help information about the command.
-d, --debug	Enables debug mode. For example, full stack traces of errors will be displayed.
-e, --eprFile <file>	Specifies an *XML* file that contains the *WS-Addressing* endpoint reference.
-s, --service <url>	Specifies the service URL.
-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.
-z, --authorization <type>	Specifies authorization type. **type** can be '**self**', '**host**', '**none**', or a string specifying the expected identity of the remote party.

Example:

```
$ wsrf-delete-property -s http://localhost:8080/wsrf/services/WidgetService \
   -k "{http://www.globus.org/namespaces/2004/06/core}WidgetKey" 123 \
   "{http://widgets.com}foo"
```

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 27. Common options

-h, --help	Displays help information about the command.
-d, --debug	Enables debug mode. For example, full stack traces of errors will be displayed.
-e, --eprFile <file>	Specifies an *XML* file that contains the *WS-Addressing* endpoint reference.
-s, --service <url>	Specifies the service URL.
-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.
-z, --authorization <type>	Specifies authorization type. **type** can be '**self**', '**host**', '**none**', or a string specifying the expected identity of the remote party.

Example:

```
$ wsn-get-current-message -s http://localhost:8080/wsrf/services/CounterService \
   -k "{http://counter.com}CounterKey" 123 \
   "{http://counter.com}Value"
```

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 28. Common options

-h, --help	Displays help information about the command.
-d, --debug	Enables debug mode. For example, full stack traces of errors will be displayed.
-e, --eprFile <file>	Specifies an *XML* file that contains the *WS-Addressing* endpoint reference.
-s, --service <url>	Specifies the service URL.
-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.
-z, --authorization <type>	Specifies authorization type. **type** can be **'self'**, **'host'**, **'none'**, or a string specifying the expected identity of the remote party.

Example:

```
$ 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 29. Common options

-h, --help	Displays help information about the command.
-d, --debug	Enables debug mode. For example, full stack traces of errors will be displayed.
-e, --eprFile <file>	Specifies an *XML* file that contains the *WS-Addressing* endpoint reference.
-s, --service <url>	Specifies the service URL.
-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.
-z, --authorization <type>	Specifies authorization type. **type** can be '**self**', '**host**', '**none**', or a string specifying the expected identity of the remote party.

Example:

```
$ wsn-resume-subscription -s http://localhost:8080/wsrf/services/SubscriptionManagerServic
   -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**.

Table 30. Common options

-h, --help	Displays help information about the command.
-d, --debug	Enables debug mode. For example, full stack traces of errors will be displayed.
-e, --eprFile <file>	Specifies an *XML* file that contains the *WS-Addressing* endpoint reference.
-s, --service <url>	Specifies the service URL.
-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.
-z, --authorization <type>	Specifies authorization type. **type** can be '**self**', '**host**', '**none**', or a string specifying the expected identity of the remote party.

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

Table 31. Command-specific options

-r, --resDescFile <file>	Specifies a file containing a resource security descriptor for the notification consumer resource.
-b, --subEpr <file>	Specifies a file to which the subscription resource EPR will be saved.

Example:

```
$ wsn-subscribe -s http://localhost:8080/wsrf/services/CounterService \
   -k "{http://counter.com}CounterKey" 123 \
   "{http://counter.com}Value"
```

Name

globus-deploy-gar -- Deploys a GAR file

`globus-deploy-gar`

Tool description

Deploys a GAR file.

Command syntax

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

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

Table 32. Options

-help	Displays help information about the command.
-debug	Enables debug mode.
-backup	Creates backup of existing configuration files (since GT 4.0.1).
-profile	Specifies the profile name under which the configuration files in the GAR will be deployed. Please see Configuration Profiles for details.
-D<property>=<value>	Passes arbitrary property-value pairs. See below for the list of currently supported properties.

Table 33. Supported property-value pairs

-Dall.scripts=true	Causes Windows and Unix launcher scripts to be generated.

Note

The **globus-deploy-gar** command always overwrites the existing configuration and non-configuration files of the service during the deployment.

Example:

```
$ globus-deploy-gar /tmp/gars/globus_wsrf_core_samples_counter.gar
```

The **globus-deploy-gar** invokes an Ant task. 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.

Note

The **globus-deploy-gar** is an Ant-based tool. If Ant is installed incorrectly this tool might not work properly. This tool will not work with Ant installed from JPackage[1].

[1] http://www.jpackage.org/

Name

globus-undeploy-gar -- Undeploys a GAR file

`globus-undeploy-gar`

Tool description

Undeploys a GAR file.

Command syntax

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

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 34. Options

-help	Displays help information about the command.
-debug	Enables debug mode.
-D<property>=<value>	Passes arbitrary property-value pairs.

Example:

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

The **globus-undeploy-gar** invokes an Ant task. 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
```

Note

The **globus-deploy-gar** is an Ant-based tool. If Ant is installed incorrectly this tool might not work properly. This tool will not work with Ant installed from JPackage[1].

[1] http://www.jpackage.org/

GT 4.0 Java WS Core Common Command Options

Name

Common Java Client Options -- list of common options across commands

Common Java Client Options

Table 35. Common options

-h, --help	Displays help information about the command.
-d, --debug	Enables debug mode. For example, full stack traces of errors will be displayed.
-e, --eprFile <file>	Specifies an *XML* file that contains the *WS-Addressing* endpoint reference.
-s, --service <url>	Specifies the service URL.
-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.
-z, --authorization <type>	Specifies authorization type. **type** can be '**self**', '**host**', '**none**', or a string specifying the expected identity of the remote party.

GT 4.0 Java WS Core Glossary

A

Apache Axis
The SOAP engine implementation used within the Globus Toolkit. See the Apache Axis website[1] for details.

C

client-config.wsdd
Axis client-side WSDD configuration file. It contains information about the type mappings, the transport and other handlers.
See Also Apache Axis, Web Services Deployment Descriptor (WSDD).

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[2] for details.
See Also jndi-config.xml, ResourceHome.

jndi-config.xml
It is a XML-based configuration file used to populate the container registry accessible via the JNDI API. See the JNDI details[3] for more information.
See Also JNDI, ResourceHome, XML.

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 GAR details[4] for more information.

O

operation provider
A reusable Java component that implements one or more web service functions. A web service can be composed of one or more operation providers. See Operation Provider[5] for more information.

R

ResourceHome
The resources are managed and discovered via *ResourceHome* implementations. The *ResourceHome* implementations can also be responsible for creating new re-

[1] http://ws.apache.org/axis/
[2] http://java.sun.com/products/jndi/tutorial/
[3] ../../common/javawscore/developer-index.html#s-javawscore-developer-JNDIDetails
[4] ../../common/javawscore/developer-index.html#s-javawscore-developer-gardetails
[5] ../../common/javawscore/developer-index.html#s-javawscore-developer-OperationProvider

sources, performing operations on a set of resources at a time, etc. *ResourceHome*s are configured in JNDI and are associated with a particular web service.
See Also JNDI.

S

server-config.wsdd

Axis server-side WSDD configuration file. It contains information about the services, the type mappings and various handlers.
See Also Apache Axis, Web Services Deployment Descriptor (WSDD).

SOAP

SOAP provides a standard, extensible, composable framework for packaging and exchanging XML messages between a service provider and a service requester. SOAP is independent of the underlying transport protocol, but is most commonly carried on HTTP. See the SOAP specifications[6] for details.

W

Web Services Deployment Descriptor (WSDD)

Axis XML-based configuration file.
See Also Apache Axis, client-config.wsdd, server-config.wsdd, XML.

WS Addressing (WSA)

The WS-Addressing specification defines transport-neutral mechanisms to address Web services and messages. Specifically, this specification defines XML elements to identify Web service endpoints and to secure end-to-end endpoint identification in messages. See the W3C WS Addressing Working Group[7] for details.

WS Notification (WSN)

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[8] for details.

WS Resource Framework (WSRF)

The WS Resource Framework (WSRF) specifications define a generic and open framework for modeling and accessing stateful resources using Web services. This framework comprises mechanisms to describe views on the state (WS-ResourceProperties), to support management of the state through properties associated with the Web service (WS-ResourceLifetime), to describe how these mechanisms are extensible to groups of Web services (WS-ServiceGroup), and to deal with faults (WS-BaseFaults). See the OASIS Web Services Notification (WSRF) TC[9] for details.

WSDL

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[10] for details.

[6] http://www.w3.org/TR/soap/
[7] http://www.w3.org/2002/ws/addr/
[8] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn
[9] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf
[10] http://www.w3.org/TR/wsdl

	See Also XML, SOAP.
WS-I Basic Profile	The WS-I Basic Profile[11] specification is a set of recommendations on how to use the different web services specifications such as SOAP, WSDL, etc. to maximize interoperability. See Also WSDL, SOAP.

X

XML	Extensible Markup Language (XML) is standard, flexible, and extensible data format. See the W3C XML site[12] for details.
XPath	XPath is a language for finding information in an XML document. XPath is used to navigate through elements and attributes in an XML document. See the XPath specification[13] for details.

[11] http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html
[12] http://www.w3.org/XML/
[13] http://www.w3.org/TR/xpath