

Java WS Core Admin Guide

DRAFT

Java WS Core Admin Guide

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 [Installing GT 4.2.0](#). Read through this guide before continuing!

DRAFT

Table of Contents

| | |
|---|----|
| 1. Building and Installing | 1 |
| 1. Building from source | 1 |
| 2. Installing Core-only binary distribution | 2 |
| 2. Configuring | 3 |
| 1. Configuration overview | 3 |
| 2. Syntax of the interface: | 3 |
| 3. Configuring an <code>/etc/init.d</code> entry for the standalone container | 11 |
| 3. Deploying | 12 |
| 1. Deploying into the Java WS Core container | 12 |
| 2. Deploying into Tomcat | 12 |
| 3. Deploying into JBoss | 17 |
| 4. Testing | 19 |
| 5. Security Considerations | 20 |
| 1. Security Considerations for Java WS Core | 20 |
| 6. Debugging | 21 |
| 1. Logging in Java WS Core | 21 |
| 7. Troubleshooting | 23 |
| 1. <code>globus-stop-container</code> fails with an authorization error | 23 |
| 2. <code>globus-start-container</code> hangs during startup | 23 |
| 3. Java WS Core Errors | 24 |
| 4. General troubleshooting information | 26 |
| 8. Usage statistics collection by the Globus Alliance | 27 |
| 1. Usage statistics sent by Java WS Core | 27 |
| Glossary | 28 |
| Index | 29 |

List of Tables

| | |
|--|----|
| 2.1. General configuration parameters | 3 |
| 2.2. Standalone/embedded container-specific configuration parameters | 4 |
| 2.3. Container Thread Parameters | 5 |
| 2.4. Default container thread pool settings | 5 |
| 2.5. Axis Standard Parameters | 7 |
| 2.6. Java WS Core Parameters | 7 |
| 2.7. ResourceHomeImpl parameters | 9 |
| 7.1. Java WS Core Errors | 25 |

DRAFT

Chapter 1. Building and Installing

Java WS Core is built and installed as part of a default [GT 4.2 installation](#). 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.

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 authorization
```

- 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.

```
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 build target.

From CVS.

```
ant all
```

From Core-only source distribution.

```
ant all -Dauthz.install=../authorization-XXX
```

For example, for GT 4.2.0, the directory will be ws-core-4.2.0 and the command would be as follows:

```
ant all -Dauthz.install=../authorization-4.2.0
```

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` - create and configure the container with a global security descriptor

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/
```

Chapter 2. Configuring

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

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

2. Syntax of the interface:

2.1. Global Configuration

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

Table 2.1. General configuration parameters

| Name | Value | Description | Comments |
|-------------------------------|----------------------------------|---|----------|
| <code>logicalHost</code> | <code><hostname></code> | This parameter specifies the hostname to use instead of the default local host. It is equivalent to setting the <code>GLOBUS_HOSTNAME</code> environment property. Can be FQDN or just hostname. | Optional |
| <code>disableDNS</code> | <code><boolean></code> | This parameter specifies whether to perform DNS lookup on the <code>logicalHost</code> parameter. By default, <code>"false"</code> is assumed (DNS lookup is performed). | Optional |
| <code>domainName</code> | <code><domain name></code> | This parameter specifies the domain name to append to the host name if the host name is not qualified by a domain. | Optional |
| <code>publishHost-Name</code> | <code><boolean></code> | This parameter specifies whether to publish the hostname or the ip address. It is only used when DNS lookups are enabled (<code>disableDNS</code> is false). | Optional |
| <code>server.id</code> | <code><string></code> | 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 <code>"<ip>-<containerPort>"</code> . In Tomcat, the server id will default to <code>"<ip>-<webApplicationName>"</code> . | Optional |

Table 2.2. Standalone/embedded container-specific configuration parameters

| Name | Value | Description | Comments |
|--------------------------|--------|---|----------|
| <i>container-Timeout</i> | <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 |
| <i>webContext</i> | <name> | This parameter specifies the context name under which the services are published under: <code>http://<host>:<port>/<webContext>/services/MyService</code> . By default "wsrf" name is used. In Tomcat, this parameter is set to the web application's name. | Optional |

2.2. Container Performance Tuning

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

2.2.1. JVM Parameters

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

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

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

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

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

2.2.2. Container Parameters

The Container parameters also play a significant role in the performance of your container. The main parameters relate to threads.

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

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

Table 2.3. Container Thread Parameters

| Name | Value | Description | Comments |
|--------------------------------------|-------|---|----------|
| <i>containerThreads</i> | <int> | This parameter controls the initial thread pool size for the container. If not set, it defaults to 1. | Optional |
| <i>containerThreadsMax</i> | <int> | This parameter sets the maximum number of threads for the container. By default it is set to 4 * the <code>containerThread</code> setting. | Optional |
| <i>containerThreadsHighWaterMark</i> | <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 <code>containerThread</code> setting. | Optional |

The default Thread settings are as follows:

Table 2.4. Default container thread pool settings

| Type | Min. threads | Max. threads |
|-------------------|--------------|--------------|
| <i>standalone</i> | 2 | 20 |
| <i>embedded</i> | 1 | 3 |

2.2.3. Client Parameters

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

make that mostly moot. Also, if you keep a connection open, you will be limiting the ability of other clients to connect. So this is a parameter that should be changed carefully, depending on your particular circumstances.

2.3. Service Configuration

2.3.1. WSDD

An example of a deployment descriptor for a CounterService:

```
<service name="CounterService" provider="Handler" use="literal" style="document"> <parameter
    value="org.globus.wsrp.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
        GetRPPProvider 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 2.5. Axis Standard Parameters

| Name | Value | Description | Comments |
|-----------------------|-------------------|--|---|
| <i>className</i> | <class> | This parameter specifies a class that implements the web service methods. | Required |
| <i>handler-Class</i> | <class> | This parameter specifies what dispatcher to use to send a request to a service method. This parameter is required if the <i>provider</i> attribute of the <i>service</i> is set to <code>Provider</code> . The default dispatcher we provide is called <i>org.globus.axis.providers.RPCProvider</i> . It enables special features such as operation providers or security support. | Recommended in our environment |
| <i>scope</i> | <value> | Scope value can be one of: <i>Request</i> (the default), <i>Application</i> , or <i>Session</i> . If <i>Request</i> scope is used, a new service object is created for each SOAP request that comes in for the service. If <i>Application</i> 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 <i>Session</i> scope is used, a new service object is created for each session-enabled client who accesses the service. <i>Note: Only Request and Application scopes are supported when used with org.globus.axis.providers.RPCProvider handlerClass.</i> | <i>Application</i> scope is recommended |
| <i>wSDLFile</i> | <path> | This parameter points to a wSDL file for the service. The wSDL file must contain the <i>wSDL:service</i> entry. The file location can be relative or absolute. A relative file location is recommended. | Required in our environment |
| <i>allowedMethods</i> | <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 2.6. Java WS Core Parameters

| Name | Value | Description | Comments |
|----------------------------|---------------------|--|----------|
| <i>loadOnStartup</i> | <boolean> | If set to <i>true</i> this parameter will cause the web service and the corresponding ResourceHome (if any) to be initialized (with proper security settings if configured) at container startup. This is useful for restarting some tasks, etc. at container startup without having to call the service. Please check Section 1.1, "How can I force my service to initialize when the container starts?" for details. | Optional |
| <i>allowedMethodsClass</i> | <class> | This parameter is similar to the <i>allowedMethods</i> 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 <i>org.globus.axis.providers.RPCProvider</i> handlerClass. | Optional |
| <i>providers</i> | <list of providers> | This parameter specifies a space separated list of provider names or class names. Please see the Operation provider support section for details. This parameter only has effect if used with <i>org.globus.axis.providers.RPCProvider</i> handlerClass. | Optional |

Please see [Custom Deployment](#)¹ for details on Axis Web Services Deployment Descriptor.

2.3.2. JNDI

An example of a JNDI configuration bit for a CounterService:

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

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

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

2.3.2.1. ResourceHomeImpl

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

¹ <http://ws.apache.org/axis/java/user-guide.html#PublishingServicesWithAxis>

Table 2.7. ResourceHomeImpl parameters

| <i>Name</i> | <i>Value</i> | <i>Description</i> | <i>Comments</i> |
|-------------------------|--------------|---|---|
| <i>resourceKey-Name</i> | <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 |
| <i>resourceKey-Type</i> | <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 <code>java.lang.String</code> |
| <i>resourceClass</i> | <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 |
| <i>sweeperDelay</i> | <long> | This parameter specifies how often the resource sweeper runs in milliseconds. | Optional. Defaults to 1 minute |
| <i>cacheLocation</i> | <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 |

2.3.2.1.1. 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](#)². That allows the JVM to automatically reclaim these resource objects, thus reducing the memory usage. Since the JVM can decide to reclaim these objects at any point, sometimes a resource object can be reclaimed between two subsequent invocations on the same resource. This for example can cause the state of the resource to be reloaded from disk on each call. To prevent the JVM from reclaiming the resource objects so quickly a cache can be setup up to hold direct references to these objects. A basic LRU (least recently used) cache implementation is provided. Other cache implementations can be used as long as they implement the *org.globus.wsrfultils.cache.Cache* interface.

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

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

```

² <http://java.sun.com/j2se/1.4.2/docs/api/java/lang/ref/SoftReference.html>

```

        <value>1000</value>
    </parameter>
</resourceParams>
</resource>

```

The *timeout* parameter (in ms) is used to specify the idle time of the resource object before it is removed from the cache. Also, the *maxSize* parameter can be used to specify the maximum size of the cache. By default the cache is configured without any size limit and 5 minutes timeout.

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

```

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

```

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

2.3.2.2. ServiceResourceHome

This implementation does not accept any special parameters.

2.4. Usage Statistics Configuration

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

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

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

2.5. Configuration Profiles

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

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

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

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

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

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

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

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

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

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

To start the container by hand run:

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

To stop the container by hand run:

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

Chapter 3. Deploying

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

1. Deploying into the 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`).

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`.

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.

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=<tomcat.dir>
```

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. To enable local invocation in Tomcat you must specify `-Dlocal.inocations=true`

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 than 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"
    encryption="true"/>
</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. The `encryption` attribute is also optional (defaults to `true` if not set).



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 [Introduction](#).

The `mode` attribute can also be set to specify the connection mode. There are two supported connection modes: `ssl` and `gsi`. The `ssl` mode indicates a regular SSL connection mode. The `gsi` mode indicates a SSL connection mode with transport-level delegation support. The `ssl` mode is the default mode if the `mode` attribute is not specified. Please note that the `gsi` mode is intended for advanced users only.

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"
encryption="true"/>

```

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. The encryption attribute is also optional (defaults to true if not set).

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 [Introduction](#).

The mode attribute can also be set to specify the connection mode. There are two supported connection modes: `ssl` and `gsi`. The `ssl` mode indicates a regular SSL connection mode. The `gsi` mode indicates a SSL connection mode with transport-level delegation support. The `ssl` mode is the default mode if the mode attribute is not specified. Please note that the `gsi` mode is intended for advanced users only.

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 of the Tomcat config file 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"
    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"
    encryption="true"/>

```

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. The `encryption` attribute is also optional (defaults to `true` if not set).

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 [Introduction](#).

The `mode` attribute can also be set to specify the connection mode. There are two supported connection modes: `ssl` and `gsi`. The `ssl` mode indicates a regular SSL connection mode. The `gsi` mode indicates a SSL connection mode with transport-level delegation support. The `ssl` mode is the default mode if the `mode` attribute is not specified. Please note that the `gsi` mode is intended for advanced users only.

2. Add a HTTPS Valve in the `<Engine name="Catalina" ... >` section of the Tomcat config file:

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

Note

It is recommend to run Tomcat with Java 1.4.2+.

2.1. web.xml configuration

You may have to edit `<tomcat.dir>/webapps/wsrif/WEB-INF/web.xml` if you are running Tomcat on a non-default port, that is if not using port 8443 (HTTPS). For example, if you run Tomcat on port 443 using HTTPS then the WSRF servlet entry should be modified to have the following `defaultProtocol` and `defaultPort` parameters:

```
<web-app> ... <servlet>
  <servlet-name>WSRFServlet</servlet-name>
  <display-name>WSRF Container Servlet</display-name>
  <servlet-class> org.globus.wsrif.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>
```

Alternatively, you can use the `setDefault`s Ant task to set the default protocol/port in the `web.xml` file:

```
$ cd $GLOBUS_LOCATION $ ant -f share/globus_wsrif_common/tomcat/tomcat.xml setDefaults \
  -Dtomcat.dir=<tomcat.dir> \
  -DdefaultPort=<port>
  -DdefaultProtocol=<protocol>
```

Also, by default the `webContext` property is set to the directory name of the web application on the file system. However, sometimes the context under which the web application is published might be different from the directory

name of the application. In such cases it is necessary to explicitly configure the published context name in the `web.xml` file. To configure the web application context name set the `webContext` parameter in `web.xml` file. For example (assuming services are published under `http://localhost:8080/foo/services`) the `webContext` should be set to:

```
<web-app> ... <servlet>
  <servlet-name>WSRFServlet</servlet-name> ...
  <init-param> <param-name>webContext</param-name>
  <param-value>foo</param-value> </init-param> ...
  <load-on-startup>true</load-on-startup> </servlet>
  ... </web-app>
```

2.2. Debugging

2.2.1. Tomcat log files

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

2.2.2. Enabling Log4J debugging

- Tomcat 4.1.x

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

2.3. Creating WAR file

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

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

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.

2.4. Deploying and undeploying services

Assuming Java WS Core is already deployed into Apache Tomcat (as described in [Deploying Java WS Core](#)), use the `globus-deploy-gar` tool with the `-tomcat <tomcat.dir>` option to deploy your GT service directly into Tomcat. Similarly,

to undeploy a service, use the `globus-undeploy-gar` tool with the `-tomcat <tomcat.dir>` option to undeploy the service from Tomcat.

Alternatively, to indirectly deploy a service into Tomcat, first deploy the service into a regular GT installation using the `globus-deploy-gar` tool and then redeploy the GT installation into Tomcat (as described in [Deploying Java WS Core](#)). Similarly, to undeploy a service, first undeploy the service from a regular GT installation using `globus-undeploy-gar` tool and then *redeploy* the GT installation into Tomcat.



Note

Some GT services may not work properly in Tomcat.

3. Deploying into JBoss

To deploy a Java WS Core installation into JBoss (version 4.0.x+) do the following:

1. Run:

```
$ cd $GLOBUS_LOCATION $ ant -f share/globus_wsrf_common/tomcat/jboss.xml
deployJBoss \ -Djboss.dir=<jboss.dir>
```

Where `<jboss.dir>` is an *absolute* path to the JBoss 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.

2. Add a HTTPS Connector and HTTPS Valve:

- a. Add a HTTPS Connector in the `<Service name="Catalina">` section of the Tomcat config file 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"
  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"
  encryption="true"/>
```

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. The `encryption` attribute is also optional (defaults to `true` if not set).

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 [Introduction](#).

The mode attribute can also be set to specify the connection mode. There are two supported connection modes: `ssl` and `gsi`. The `ssl` mode indicates a regular SSL connection mode. The `gsi` mode indicates a SSL connection mode with transport-level delegation support. The `ssl` mode is the default mode if the mode attribute is not specified. Please note that the `gsi` mode is intended for advanced users only.

- b. Add a HTTPS Valve in the `<Engine name="Catalina" ... >` section of the Tomcat config file:

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



Note

JBoss 4.0.x+ installation with embedded Tomcat is required.

The Tomcat configuration file should be under `<jboss.dir>/default/deploy/jbossweb-tomcat55.sar/server.xml`.

Chapter 4. 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](#) for more information on running the tests and the testing infrastructure.

Chapter 5. Security Considerations

1. Security Considerations for Java WS Core

1.1. 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.

1.2. 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.

1.3. Invocation of non-public service functions

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

1. Define all service methods in your service class as either `private` or `protected`.
2. Configure appropriate *allowedMethods* or *allowedMethodsClass* parameter in the service deployment descriptor (please see [Configuring Java WS Core](#) for details).

Chapter 6. Debugging

1. Logging in Java WS Core

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

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

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

1.1. Development Logging in Java WS Core

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

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

1.1.1. Configuring server side developer logs

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

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

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

1.1.2. Configuring client side developer logs

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

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

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

³ <http://logging.apache.org/log4j/>

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

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

1.2. Configuring system administration logs

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

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

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

1.3. Sample log file

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

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

Chapter 7. Troubleshooting

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

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_P
  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 [WS Authentication & Authorization](#) for details.

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.

3. Java WS Core Errors

DRAFT

Table 7.1. Java WS Core Errors

| Error Code | Definition |
|---|---|
| Failed to acquire notification consumer home instance from registry | Caused by <code>javax.naming.NameNotFoundException</code> : Name <code>services</code> is not bound in |
| The WS-Addressing 'To' request header is missing | This warning is logged by the container if the request did not contain the necessary <i>WS-Addressing</i> headers, those headers at all or is somehow misconfigured. |
| <code>java.io.IOException: Token length X > 33554432</code> | If you see this error in the container log, it usually means you are trying to connect to HTTPS server using <code>https</code> specifies 8443 as a port number and <code>http</code> as the protocol name. |
| <code>java.lang.NoSuchFieldError: DOCUMENT</code> | This error usually indicates a mismatch between the version of Apache Axis that the code was compiled with and the version currently running with. |
| <code>org.globus.wsrfl.InvalidResourceKeyException: Argument key is null / Resource key is missing</code> | These errors usually indicate that a resource key was not passed with the request or that an invalid resource key was used (the element QName of the resource key did not match what the service expected). |
| Unable to connect to localhost:xxx | Cannot resolve localhost. The machine's <code>/etc/hosts</code> isn't set up correctly and/or you do not have DNS for localhost. |
| <code>org.globus.common.ChainedIOException: Failed to initialize security context</code> | This may indicate that the user's proxy is invalid. |
| Error: <code>org.xml.sax.SAXException: Unregistered type: class xxx</code> | This may indicate that an Axis generated XML type, defined by the WS RLS XSD, was not properly registered upon deployment without intervention by the user, sometimes they do not. |
| No socket factory for 'https' protocol | When a client fails with the following exception: <pre>java.io.IOException: No socket factory for 'https' protocol at org.apache.axis.transport.http.HTTPSender.getSocket(HTTPSender.java:100) org.apache.axis.transport.http.HTTPSender.writeToSocket(HTTPSender.java:100) org.apache.axis.transport.http.HTTPSender.invoke(HTTPSender.java:100)</pre> <p>FIXME - it may have happened because...</p> |

| Error Code | Definition |
|---|---|
| No client transport named 'https' found | <p>When a client fails with the following exception:</p> <pre>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)</pre> <p>The client is most likely loading an incorrect <code>client-config.wsdd</code> configuration file.</p> |
| ConcurrentModificationException in Tomcat 5.0.x | <p>If the following exception is visible in the Tomcat logs at startup, it might cause the HTTPSValve to fail:</p> <pre>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.</pre> <p>The HTTPSValve might fail with the following exception:</p> <pre>java.lang.NullPointerException at org.apache.coyote.tomcat5.CoyoteRequest.setAttribute(CoyoteRequestFacade.java:100) at org.apache.coyote.tomcat5.CoyoteRequestFacade.setAttribute(CoyoteRequestFacade.java:100) at org.globus.tomcat.coyote.valves.HTTPSValve.expose(HTTPSVAlve.java:100)</pre> <p>These exceptions will prevent the transport security from working properly in Tomcat.</p> |
| java.net.SocketException: Invalid argument or cannot assign requested address | <p>FIXME - what causes this?</p> |
| GAR deploy/undeploy fails with container is running error | <p>A GAR file can only be deployed or undeployed locally while the container is off. However, GAR deployments fail with this error even if the container is off. This usually happens if the container has crashed or was stopped from cleaning up its state files.</p> |

4. General troubleshooting information

- In general, if you want to investigate a problem on your own please see [Debugging](#) for details on how to turn on debugging.
- Most of the command line clients have a `-debug` option that will display more detailed error messages, including the error stack traces.
- [Search the mailing lists](#)¹ such as gt-user@globus.org² or jwscore-user@globus.org³ (before posting a message).
- If you think you have found a bug please report it in our [Bugzilla](#)⁴ system. Please include as much as detail about the problem as possible.

¹ <http://www.globus.org/email-archive-search.php>

² <mailto:gt-user@globus.org>

³ <mailto:jwscore-user@globus.org>

⁴ <http://bugzilla.globus.org/bugzilla/>

Chapter 8. Usage statistics collection by the Globus Alliance

1. Usage statistics sent by Java WS Core

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 deployed 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
 - container uptime

If you wish to disable this feature, please see the "Usage Statistics Configuration" section of [Section 2.4, "Usage Statistics Configuration"](#) for instructions.

Also, please see our [policy statement](#)¹ on the collection of usage statistics.

¹ http://www.globus.org/toolkit/docs/4.2/4.2.0/Usage_Stats.html

Glossary

G

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

J

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

R

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

S

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

W

Web Services Addressing (WSA) The WS-Addressing specification defines transport-neutral mechanisms to address web services and messages. Specifically, it defines XML elements to identify web service endpoints and to secure end-to-end endpoint identification in messages. See the [W3C WS Addressing Working Group](http://www.w3.org/2002/ws/addr/)¹⁴ for details.

Web Services Description Language (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](http://www.w3.org/TR/wsdl)¹⁵ for details.

¹⁴ <http://www.w3.org/2002/ws/addr/>

¹⁵ <http://www.w3.org/TR/wsdl>

Index

U

usage statistics, 27

B

building and installing

- Core-only binary distribution, 2
- from source, 1
- general instructions, 1

C

configuring, 3

- /etc/init.d for the standalone container, 11
- global, 3
- overview, 3
- profiles, 11
- service, 6
 - JNDI, 8
 - WSDD, 6
- usage statistics, 10

D

debugging

- logging, 21

deploying your installation

- into JBoss, 17
- into standalone container, 12
- into Tomcat, 12

E

errors, 24

I

installing

- Core-only binary distribution, 2
- from source, 1
- general instructions, 1

L

logging

- CEDPS-compliant, 21
- debugging, 21

S

security considerations, 20

T

testing

- your installation, 19

troubleshooting

- for sys admins, 23