

GT 4.2.1 CAS Admin Guide

GT 4.2.1 CAS Admin Guide

Introduction

This guide contains advanced configuration information for system administrators working with the Community Authorization Service (CAS). It provides references to information on procedures typically performed by system administrators, including installing, 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 4.2.1](#). Read through this guide before continuing!

Note: Typically a single CAS server is run per VO and multiple client installations are done. This document contains information about deploying a CAS server and is not needed for a CAS client installation. Please refer to the documentation for [CAS client install \[1\]](#).

Table of Contents

1. Building and Installing	1
2. Configuring	2
1. Configuration overview	2
2. Loading the CAS service at start up	2
3. Configuring the VO Description	3
4. Configuring the maximum assertion lifetime	3
5. Configuring database backend	3
6. Configuring security descriptor	3
7. Configuring with a GridFTP Server	4
8. Configuring CAS to manage policy for web service.	4
9. CAS auto-registration with default WS MDS Index Service	4
10. Registering CAS manually with default WS MDS Index Service	6
3. Deploying	7
1. Obtaining credentials for the CAS service	7
2. Database installation and configuration	7
3. Deploying into Tomcat	9
4. Setting up CAS for GridFTP	10
1. Overview	10
2. Files and directories in CAS	10
3. Configuring the CAS server to work with the GridFTP server	11
4. Transferring files using CAS and GridFTP	14
5. Managing policy	14
6. Authorization Enforcement at the GridFTP Server	15
5. Testing	16
1. Testing the back end database module	16
2. Testing the CAS service module	17
6. Example of CAS Server Administration	20
1. Adding a user group	20
2. Adding a trust anchor	21
3. Adding users	21
4. Adding users to a user group	21
5. Adding a new FTP server	21
6. Creating an object group	22
7. Adding members to an object group	22
8. Adding service types	22
9. Adding action mappings	23
10. Granting permissions	23
7. Example of using CAS Service for Web Service Policy Management	24
1. Example of using CAS Service as Authorization Service	24
2. Example of using CAS Service as Local PDP	26
8. Security Considerations	30
1. Security Considerations for CAS	30
9. Debugging	31
1. Logging in Java WS Core	31
10. Troubleshooting	33
1. Error Messages	34
2. Credential Troubleshooting	35
Glossary	38

List of Tables

2.1. Database parameters	3
2.2. Mapping from web services object to CAS object	4
3.1. Command line options	8
5.1. Test database properties	16
5.2. Test properties	17
7.1. Entity Mapping	24
7.2. Operation Details	27
10.1. WS A&A Authorization Framework Error Messages	34
10.2. Credential Errors	36

Chapter 1. Building and Installing

The CAS server and client are built and installed as part of a default GT 4.0 installation. For basic installation instructions, refer to the [Installing GT 4.2.1](#). No extra installation steps are required for this component.

The CAS client can be installed by itself. Please refer to *FILL ME*(*olink to top level install guide for installing a given package*)

Chapter 2. Configuring

1. Configuration overview

The CAS service can be configured with the following :

- server start up configuration
- a description of the VO the CAS service serves
- the maximum lifetime of the assertions it can issue
- information about the back end database it uses. Any database with a JDBC driver and reasonable SQL support can be used. The schema that works with Derby database, MySQL and PostGres is distributed and can be found at `$GLOBUS_LOCATION/etc/globus_cas_service/casDbSchema`.
- the security settings of the service and can be modified in the security descriptor associated with the CAS service. It allows for configuring the credentials that will be used by the service, the type of authentication and message protection required as well as the authorization mechanism.

2. Loading the CAS service at start up

By default, the CAS service is not loaded at start up. To change this behavior, uncomment the *loadOnStartup* property set in `$GLOBUS_LOCATION/etc/globus_cas_service/server-config.wsdd` as shown below.

Once the *loadOnStartup* property is uncommented, the following happens at start up:

1. The CAS service is loaded.
2. The database connection pool is initialized.
3. The service registers itself to the default MDS Index Service.

```
<service name="CASService" provider="Handler" use="literal"
  style="document">
  <!-- Uncomment if the service needs to be initialized at startup -->
  <parameter name="loadOnStartup" value="true"/>
  <parameter name="allowedMethodsClass"
  value="org.globus.cas.CASPortType"/>
  .
  .
  .
</service>
```

3. Configuring the VO Description

To change the VO description, set the parameter `voDescription` in `$GLOBUS_LOCATION/etc/globus_cas_service/jndi-config.xml` to the desired values.

4. Configuring the maximum assertion lifetime

To change the maximum assertion lifetime set the parameters `maxAssertionLifetime` in `$GLOBUS_LOCATION/etc/globus_cas_service/jndi-config.xml` to the desired values.

5. Configuring database backend

To alter the configuration of the database back end edit the `databaseConfiguration` section of `$GLOBUS_LOCATION/etc/globus_cas_service/jndi-config.xml` as described below. If you are using the default Derby installation, the only parameter to change is the `connectionURL` to replace `GLOBUS_LOCATION` with the actual location of your toolkit installation.

Table 2.1. Database parameters

<code>driver</code>	The JDBC driver to be used
<code>connectionURL</code>	The JDBC connection url to be used when connecting to the database
<code>userName</code>	The user name to connect to the database as
<code>password</code>	The corresponding database password
<code>activeConnections</code>	The maximum number of active connections at any given instance
<code>onExhaustAction</code>	The action to perform when the connection pool is exhausted. If value is 0 then fail, if 1 then block and if 2 then grow the pool (get more connections)
<code>maxWait</code>	The maximum time in milliseconds that the pool will wait for a connection to be returned
<code>idleConnections</code>	The maximum number of idle connections at any given time

6. Configuring security descriptor

By default, the following security configuration is installed:

- Credentials are determined by the container level security descriptor. If there is no container level security descriptor or if it does not specify which credentials to use then default credentials are used.
- Authentication and message integrity protection is enforced for all methods except `queryResourceProperties` and `getResourceProperty`. This means that you may use any of GSI *Transport*, GSI Secure Message or GSI Secure Conversation when interacting with the CAS service.
- The standard authorization framework is not used for authorization. Instead the the service uses the back end database to determine if the call is permitted.

To alter the security descriptor configuration refer to [Security Descriptors](#). The file to be changed is `$GLOBUS_LOCATION/etc/globus_cas_service/security-config.xml`.

 **Note**

Changing required authentication and authorization methods will require matching changes to the clients that contact this service.

 **Important**

If the service is configured to use GSI Secure Transport, then container credentials are used for the handshake, irrespective of whether service level credentials are specified.

7. Configuring with a GridFTP Server

CAS is used to administer access rights to files and directories and the GridFTP server can be configured to enforce those rights.

For detailed information about configuring CAS for use with a GridFTP server, see [How to Set up CAS with GridFTP](#).

8. Configuring CAS to manage policy for web service.

The CAS server can be used to administer rights for access to web services. The mapping from CAS objects to the web service resource is shown on this table:

Table 2.2. Mapping from web services object to CAS object

Object	EPR of WS resource as string. The OGSA-AuthZ specification defines how a EPR can be represented as a string and a utility for such is provided at <code>org.globus.wsrf.impl.security.EPRUtil</code> .
Object namespace	The object namespace is used to get both a comparison algorithm and the basename. For web services policy we need exact comparison and also don't have any base name. An implicit namespace <code>casDefaultNs</code> with the required properties is added to the service.
Service type	The OGSA-AuthZ specification defines a service type to use for web services operation as "http://www.gridforum.org/namespaces/2003/06/ogsa-authorization/saml/action/operation" This is defined as a constant in <code>org.globus.wsrf.impl.security.authorization.SAMLAuthorizationConstants</code> and is added implicitly.
Action	This is the actual operation on the webservice. For example method "add" on Counter Service.

An example scenario is described [here](#).

9. CAS auto-registration with default WS MDS Index Service

With a default GT 4.0.1 installation, CAS is automatically registered with the default [WS MDS Index Service](#) running in the same container for monitoring and discovery purposes.

 **Note**

If you are using GT 4.0.0, we strongly recommend upgrading to 4.0.1 to take advantage of this capability.

However, if must use GT 4.0.0, or if this registration was turned off and you want to turn it back on, this is how it is configured:

There is a jndi resource defined in `$GLOBUS_LOCATION/etc/globus_cas_service/jndi-config.xml` as follows :

```
<resource name="mdsConfiguration"
  type="org.globus.wsrfl.impl.servicegroup.client.MDSConfiguration">
  <resourceParams>
  <parameter>
  <name>reg</name>
  <value>true</value>
  </parameter>
  <parameter>
  <name>factory</name>
  <value>org.globus.wsrfl.jndi.BeanFactory</value>
  </parameter>
  </resourceParams>
</resource>
```

To configure the automatic registration of CAS to the default WS MDS Index Service, change the value of the parameter `<reg>` as follows:

- `true` turns on auto-registration; this is the default in GT 4.0.1.
- `false` turns off auto-registration; this is the default in GT 4.0.0.

9.1. Configuring resource properties

By default, the `VoDescription` resource property (which describes the virtual organization relevant to the CAS Service) is sent to the default Index Service.

You can configure which resource properties are sent in the registration.xml file, `$GLOBUS_LOCATION/etc/globus_cas_service/registration.xml`. The following is the relevant section of the file:

```
<Content xsi:type="agg:AggregatorContent "
  xmlns:agg="http://mds.globus.org/aggregator/types">
  <agg:AggregatorConfig xsi:type="agg:AggregatorConfig">
  <agg:GetResourcePropertyPollType
  xmlns:cas="http://www.globus.org/07/2004/cas">
  <!-- Specifies that the index should refresh information
  every 8 hours (28800000ms) -->
  <agg:PollIntervalMillis>28800000</agg:PollIntervalMillis>
```

```
<!-- specifies that all Resource Properties should be
collected from the RFT factory -->

<agg:ResourcePropertyName>cas:VoDescription</agg:ResourcePropertyName>

</agg:GetResourcePropertyPollType>
</agg:AggregatorConfig>
<agg:AggregatorData/>
</Content>
```

10. Registering CAS manually with default WS MDS Index Service

If a third party needs to register an CAS service manually, see [Registering with mds-servicegroup-add](#) in the WS MDS Aggregator Framework documentation.

Chapter 3. Deploying

The CAS service is deployed as a part of a standard toolkit installation. Please refer to the [Installing GT 4.2.1](#) for details. Other than the steps described in the above guide, the following are needed to deploy the CAS service.

1. Obtaining credentials for the CAS service

The CAS service can run with its own service specific credentials. Instructions for obtaining *service credentials* may be found [here](#)¹.

The standard administrator clients that come with the distribution by default use identity authorization to authorize the service they are running against (and expect that the CAS service has credentials that have the FQDN of the host the server is running on and the service name "cas" as part of DN). Command line options can be used to specify the identity of the CAS service, if the default identity is not used. The command in the above mentioned [web page](#)² may be altered as follows to get credentials for the CAS server:

```
casadmin$ grid-cert-request -service cas -host FQDN
```

The certificate and *private key* are typically placed in `/etc/grid-security/cas-cert.pem` and `/etc/grid-security/cas-key.pem`, respectively. In this document the locations of certificate and key files are referred to as `CAS_CERT_FILE` and `CAS_KEY_FILE`, respectively. The subject name in these credentials is expected by CAS clients by default.

2. Database installation and configuration

CAS uses a back end database to store all user data. This section briefly describes the installation of such a database and the [creation of the database](#) using the schema required by the CAS back end.

2.1. Installing the database

While any database with a JDBC driver and support for a reasonable set of SQL may be used, Derby has been used for development and testing. The driver for the same is included in the distribution. A default Derby database is created for CAS during installation and the CAS server uses the database in the embedded mode. One set of CAS tests also use the database in the embedded mode and another set uses in the network mode.

Note

Any JDBC compliant database might be used as backend. If a different database is used, the corresponding driver should be added to `$GLOBUS_LOCATION/lib`.

2.2. Creating the CAS database

By default a Derby database is created during deploy time at `$GLOBUS_LOCATION/var`. But schema for using PostGres or MySQL as backend is provided at `$GLOBUS_LOCATION/etc/globus_cas_service/casDbSchema/cas_pgsql_database_schema.sql` or `$GLOBUS_LOCATION/etc/globus_cas_service/casDbSchema/cas_mysql_database_schema.sql`. Please refer to PostGres or MySQL documentation for steps involved in creating the database.

¹ <http://www.globus.org/toolkit/docs/2.4/admin/guide-verify.html#ldapcert>

² <http://www.globus.org/toolkit/docs/2.4/admin/guide-verify.html#ldapcert>

 **Note**

This documentation assumes the database name to be `casDatabase`

2.3. Bootstrapping the CAS database

The CAS database needs to be initialized with data specific to CAS and information about a super user to allow bootstrapping of CAS operations. The command line script `cas-server-bootstrap` can be used to achieve this.

```
cas-server-bootstrap [<options>] -d <dbPropFile> [ -implicit | -b <bootstrapFile> ]
```

Table 3.1. Command line options

<code>-help</code>	Prints the help message.
<code>-debug</code>	Runs the script with debug trace.
<code>-d dbProperties-File</code>	File name with database properties as follows: <code>dbDriver=database driver name</code> <code>dbConnectionURL=database connection URL</code> <code>dbUserName=Username to access database</code> <code>dbPassword=Password for the above username</code>
<code>-b bootstrapFile</code>	This option populates the database with super user data and points to a file with data to use for bootstrapping the database. A template file for this can be found at <code>\$GLOBUS_LOCATION/share/globus_cas_service/bootstrapTemplate</code> and a sample file can be found at <code>\$GLOBUS_LOCATION/share/globus_cas_service/bootstrapSample</code> .
<code>-implicit</code>	Populates the database with: a) CAS server implicit data—this adds the CAS server itself as a CAS object and implicit service/actions like enrolling users, objects and so on; and b) service/action and namespace relevant to FTP like read, write and so on.

Sample bootstrap command:

To bootstrap the CAS database with both implicit and user data the following command can be used. Prior to running the command, the following files need to be created with appropriate values filled in. The values shown here are for default Derby database set up. If using any other database configuration, please fill in or uncomment corresponding values as appropriate.

- `$GLOBUS_LOCATION/share/globus_cas_service/casDbProperties`

```
# Database driver to use
dbDriver=org.apache.derby.jdbc.EmbeddedDriver
# Connection URL to database
dbConnectionURL=jdbc:derby:casDatabase
# Database user name
dbUsername=casAdmin
```

```
# Database password
dbPassword=foobar
```

- `$GLOBUS_LOCATION/share/globus_cas_service/bootstrapProperties`. Copy the template file `$GLOBUS_LOCATION/share/globus_cas_service/bootstrapTemplate` as `$GLOBUS_LOCATION/share/globus_cas_service/bootstrapProperties` and fill in the values as described below. Refer `$GLOBUS_LOCATION/share/globus_cas_service/bootstrapSample` for a sample.

```
# This file is used to populate the backend CAS database with a super user.
# Nick name for trust anchor for super user
ta-name=defaultTrustAnchor
# Trust Anchor authentication method, typically X509 Certificates
ta-authMethod=X509
# Trust Anchor authentication data, typically X509 DN
ta-authData=/C=US/O=Globus/CN=Default CA
# Super user nickname
user-name=superUser
# Super user subject name, typically DN
user-subject=/O=Grid/O=Globus/OU=something/CN=suUser
# Super user group name
userGroupname=superUserGroup
```

- Command to run:

```
casadmin$ cd $GLOBUS_LOCATION

casadmin$ bin/cas-server-bootstrap \

-d share/globus_cas_service/casDbProperties \

-implicit -b \ share/globus_cas_service/bootstrapProperties
```

Once the database has been created the CAS service needs to be configured to use it as described [here](#).

3. Deploying into Tomcat

CAS has been tested to work without any additional setup when deployed into Tomcat. Please follow these [basic instructions](#) to deploy GT4 services into Tomcat. Note that the Java WS Core module needs to be built and configured as described in the previous sections.

Chapter 4. How to Set Up CAS to Use with GridFTP

This document is intended to be used with the [CAS](#) and [GridFTP](#) documents, to configure and use CAS with GridFTP.

1. Overview

CAS is used to administer access rights to files and directories and GridFTP server can be configured to enforce those rights. The following is an overview of the process:

1. A CAS administrator sets up rights on the CAS server.
2. A user retrieves those rights in the form of a signed assertion and presents it to the GridFTP server at the time of access.
3. If the assertion presented has the necessary permissions and the GridFTP server trusts the CAS server that issued it, access is allowed.

In a default Globus Toolkit installation, CAS is installed with a set of credentials associated with the CAS server. A CAS administrator has all rights on the CAS server and manages users. To be used with GridFTP, the administrator needs to add users, add the files and directories he wants protected and grant specific rights to the files.

The user then retrieves a CAS assertion from the server, for either specific files/directories or all his access rights stored in CAS. This is done using the client side command, [cas-proxy-init](#), which embeds the assertion in the user's proxy. Another client side command, [cas-wrap](#) can then be used to run the GridFTP command, [globus-url-copy](#), to ensure that the proxy is used.

Details about the CA who issues these credentials needs to be added to the CAS server as a trust anchor. Each user has a nick name, a subject DN and is associated with a trust anchor. A particular user is designated as superuser, so has permissions to add users and manager rights. This can be set up using the [bootstrap](#) step.

Users who are not administrators can be added using [admin command line tools](#). Permissions in a CAS database are granted to user groups, rather than users. So a user group needs to be created and users added to it prior to assigning permissions.

2. Files and directories in CAS

Files and directories are "objects" in CAS. Each object has a name and a namespace associated with it. The namespace determines the scope of the object and also the comparison method to use with object names. A predefined namespace, called FTPDirectoryTree is loaded in the CAS database upon bootstrap and this is the namespace expected for GridFTP objects.

For example, if permission on file "some_file_path" on host "somehost.edu" needs to be managed, it needs to be added as an object, with name "ftp://somehost.edu/some_file_path" and namespace "FTPDirectoryTree". This object "ftp://somehost.edu/some_file_path" will be recognized by the CAS-enabled GridFTP server at "somehost.edu" as referring to the file named "some_file_path". An object in the FTPDirectoryTree namespace with the name "ftp://somehost.edu/some_directory_path/*" will be recognized by the CAS-enabled GridFTP server at "somehost.edu" as referring to all files and directories under "some_directory_path".

In some cases, it may be desirable to have a GridFTP server recognize CAS assertions that use a hostname other than the server's fully qualified domain name. Starting the GridFTP server with the option "-H otherhost" will cause the GridFTP server to recognize objects with names that start with "ftp://otherhost/" instead.

The actions that are allowed on a file and a directory are listed in the following table:

Action	Result for a file	Result for a directory
read	Gives permission to read the file.	Gives permission to chdir to the directory.
lookup	Gives the right to get Unix stat() information.	Gives the right to chdir to and to list the contents of the directory.
write	Allows modification of an existing file.	Gives the right to chdir to the directory.
create	Allows creation of the file if it does not exist.	Allows creation of the directory if it does not exist and gives the right to chdir to the directory if it does exist.
delete	Allows deletion of the file.	Allows deletion of the directory, if empty; also gives the right to chdir to the directory.
chdir	Does not apply.	Allows making the directory the current default directory.

These actions are predefined and loaded onto CAS database when bootstrap is done. Apart from these actions on files/directories, there is another action "authz_assert" which is used to override the existing assertions (either from proxy or from a previous "authz_assert" command) with the assertions received over the GridFTP control channel. FTP clients can use the command "SITE AUTHZ_ASSERT" to send assertions to the GridFTP server.

The following is a summary of supported FTP commands and the permissions they require:

Typical Client Command	FTP Protocol Command	Rights Required
get	RETR	read
put	STOR	write, if file exists; create, if file does not exist
delete	DELE	delete
ls	LIST	lookup
chdir	CWD	any of: chdir, lookup, read, write, create, or delete
mkdir	MKD	create
rmdir	RMD	delete
rename	RNFR / RNTD	read and delete on old file; write on new file, if it exists, create on new file, if it does not exist

3. Configuring the CAS server to work with the GridFTP server

3.1. Step 1: Set up the CAS server.

This must be done by user "casAdmin".

3.2. Step 2: Run the CAS server with host credentials

Run the CAS server with host credentials `/O=Grid/OU=test/CN=foo.bar.edu`.



Note

The CA that issues these credentials should be trusted by the GridFTP server. Setting up of trusted CA is described [here](#)

3.3. Step 3: Enable CAS support in the GridFTP server

The GridFTP Server reads two files (`gsi-authz.conf` and `gsi-gaa.conf`) to determine how to perform certain authorization and mapping functions. If these files are not present (as is the case after a standard Globus Toolkit installation), the GridFTP server will not support CAS authorization (that is, the GridFTP server will ignore the CAS policy assertions in the user's credential and determine the user's permissions based solely on the user's identity).

Run **[`setup-globus-gaa-authz-callout`]** to create `gsi-authz.conf` and `gsi-gaa.conf` files that will cause the GridFTP server to honor CAS policy assertions. There are two ways to run this command:

1. To create the config files in the directory `/etc/grid-security` (where the GridFTP server looks for them by default), run the following as root:

```
$GLOBUS_LOCATION/setup/globus/setup-globus-gaa-authz-callout
```

2. To create the config files to another directory, run:

```
$GLOBUS_LOCATION/setup/globus/setup-globus-gaa-authz-callout -d mydir
```

Where 'mydir' is the path to the desired directory. You then must make sure the GridFTP server finds these files by setting the following environment variable *before* starting the GridFTP server:

- Set `GSI_AUTHZ_CONF` to `mydir/gsi-authz.conf`.
- Set `GSI_GAA_CONF` to `mydir/gsi-gaa.conf`.

By default, `setup-globus-gaa-authz-callout` will not overwrite an existing configuration file. Use the following options to overwrite existing GridFTP config files:

- Use the `-force` option to overwrite an existing `gsi-authz.conf` file
- Use the `-overwrite_gaa_config` option to overwrite an existing `gsi-gaa.conf` file.

3.4. Step 4: Configure the GridFTP server to trust the CAS server

The previous step configured the GridFTP server to understand CAS credentials. However, the GridFTP server will not allow a user authenticating with a CAS credential to perform any action that it would not allow the CAS server itself to perform. To configure the GridFTP server to trust a particular CAS server:

1. Create a local user account corresponding to the CAS server.
2. Use file permissions to allow that user account to have the desired level of file access.

3. Create a gridmap entry that maps the CAS server's distinguished name to that local account.

3.5. Step 5: Add a CAS administrator

Add a CAS admin, we'll use the CAS nickname, "casAdmin", and DN "/O=Grid/OU=test/CN=CAS Administrator". This credential is issued by a CA with certificate "/O=Grid/OU=test/CN=CA"

3.5.1. Using bootstrap to add a CAS administrator

This can be setup during [bootstrap](#) and below is the bootstrap file for the above scenario.

```
ta-name=defaultTrustAnchor
ta-authMethod=X509
ta-authData=/O=Grid/OU=test/CN=CA
user-name=superUser
user-subject=/O=Grid/OU=test/CN=CAS Administrator
userGroupname=superUserGroup
```

3.6. Step 6: Add a CAS user group

Since permissions are only on user groups, we need to add a user group, for example "readGroup". This can be done using [cas-group-admin](#).

```
cas-group-admin user create superUserGroup readGroup
```



Note

The superUserGroup here determined the user group that has permissions to manipulate the group that is being created i.e readGroup.

3.7. Step 7: Add a CAS user

Add the user who needs access to files. For example, add User1, with DN "/O=Grid/OU=test/CN=User 1" issued by the CA in [Step 4], using [cas-enroll]:

```
cas-enroll user superUserGroup User1 "/O=Grid/OU=test/CN=User 1" defaultTrustAnchor
```



Note

The superUserGroup here determined the user group that has permissions to manipulate the user that is being created i.e User1.

3.8. Step 8: Add user to the user group

To add the CAS user to the CAS user group, use [cas-group-add-entry](#):

```
cas-group-add-entry user readGroup User1
```

3.9. Step 9: Add CAS object

Add the file (on which you want to set permissions) as a CAS object with the name "ftp://foo.bar.edu/tmp/file1" and namespace "FTPDirectoryTree":

```
cas-enroll object superUserGroup "ftp://foo.bar.edu/tmp/file1" "FTPDirectoryTree"
```

 **Note**

The `superUserGroup` here determined the user group that has permissions to manipulate the object that is being created.

3.9.1. Giving permissions to all files in a directory

To give permissions to all files in a directory, create the CAS object with a wild card `"*"`. For example, object name `"ftp://foo.bar.edu/tmp/admin/*"`. If permission is given on this object, all files in that directory are affected.

For example, if you create a new user group called `"adminGroup"`, you give read permission to all users in that group as follows, any user in `"adminGroup"` can read any file in `"/tmp/admin"` on that server.

```
cas-rights-admin grant adminGroup object FTPDirectoryTree
    "ftp://foo.bar.edu/tmp/admin/*" serviceAction file read
```

3.10. Step 10: Add permission for users

Add permission for users in `"readGroup"` to be able to read object added in the previous step.

```
cas-rights-admin grant readGroup object FTPDirectoryTree
    "ftp://foo.bar.edu/tmp/file1" serviceAction file read
```

Now any user added to `readGroup` can read the file (that was added in Step 9).

4. Transferring files using CAS and GridFTP

Once the CAS server has been configured (as above), if `User1` wants to transfer the file `"/tmp/file1"` on `"ftp://foo.bar.edu"`, `User1` performs the following two steps:

1. Runs **cas-proxy-init** to get the CAS assertion:

```
cas-proxy-init -p casProxy
```

 **Note**

This gets the assertion from the CAS server, generates a proxy with the assertion and writes it out to `"casProxy"`.

2. Runs **cas-wrap** to transfer the file:

```
cas-wrap -p casProxy globus-url-copy gsiftp://somehost.edu/some_file_path
    file:///some_file_path
```

5. Managing policy

The CAS administrator can create groups of users and groups of objects to ease policy management.

Also, if the administrator wants to provide a set of rights, such as read and lookup, the administrator can create a service action group with these action as members. User groups can then be provided with access to all actions in the group.

Similarly the administrator can create groups of objects, which would be a group of GridFTP server and files on it, and grant access rights to users.

6. Authorization Enforcement at the GridFTP Server

The following describes how the GridFTP server processes a file transfer using CAS:

1. During the authentication phase, GridFTP server extracts the CAS assertion in the proxy credentials and stores it.
2. If the GridFTP server receives a CAS assertion over the control channel, it overwrites the old assertion (if any) with the new one.
3. Upon receiving a file transfer request, the GridFTP server looks for a CAS assertion.
4. If the assertion is present, checks whether the file being accessed has permission. If assertion has permission for the file, the access is allowed. Otherwise an error is thrown.
5. If no CAS assertion is found, the presence of user DN is checked in the gridmap file. If DN exists, then access is allowed. Otherwise an error is thrown.

Chapter 5. Testing

CAS has two sets of tests, one for the back end database access module and another set to test the service itself. To install both tests, install the CAS test package (*gt4-cas-delegation-test-\$version;-src_bundle.tar.gz*) using GPT. *FILLME: instructions* into *GLOBUS_LOCATION*.

Assumptions:

- A back end database has been set up and configured. By default Derby is set up and configured.
- The CAS service and tests are installed in *\$GLOBUS_LOCATION*.
- The sample commands assume:
 1. The container is started up on localhost and port 8443.
 2. The default Derby database configuration is used, i.e database name is *casDatabase* and user is *casAdmin*

1. Testing the back end database module

1. Run:

```
cd $GLOBUS_LOCATION
```

2. Populate the file *etc/globus_cas_unit_test/casTestProperties* with the database configuration information shown in the following table.

Table 5.1. Test database properties

dbDriver	The JDBC driver to be used
dbConnectionURL	The JDBC connection url to be used to connect to the database
dbUsername	The user name to use when connecting to the database
dbPassword	The password corresponding to the user name

If default Derby configuration is required,

- a. Uncomment the lines in *etc/globus_cas_unit_test/casTestProperties* following the comment *# Uncomment below for Derby Embedded Driver.*
 - b. Edit *dbConnectionURL* to replace *GLOBUS_LOCATION* with path to your installation. For example, if your *GLOBUS_LOCATION* is */temp/globus*, then the property should look like *dbConnectionURL=jd-bc:derby:/temp/globus/var/casDatabase*
 - c. Comment out all other occurrences of these properties
3. The database needs to be empty for these tests to work and will be deleted by this target. Run:

```
set GLOBUS_LOCATION=C:\globus
ant -f share/globus_cas_unit_test/cas-test-build.xml testDatabase
```

4. Test reports are placed in `$GLOBUS_LOCATION/share/globus_cas_unit_test/cas-test-reports`.

Important

The database bootstrap needs to be done again for the server to be ready to receive client requests.

2. Testing the CAS service module

These tests can be set up so as to be able to test multiple user scenarios or they can be configured to run as just a single identity. The first set of tests are used to test admin functionality and set up the database for a second user. As the second user the permissions and queries are tested to ensure that the setup worked.

As with the previous any database backend can be used for testing. If the default Derby database is used, it needs to be configured in network mode since these tests require that both the tested CAS server and the tests use the database. Two files need to be modified for configuring these tests and they are described below.

1. The `etc/globus_cas_unit_test/casTestProperties` needs to be configured with appropriate database information and service information.

The database information is configured similar to previous test, but for the default Derby installation, the network driver needs to be used.

- a. Uncomment the lines in `etc/globus_cas_unit_test/casTestProperties` following the comment *# Uncomment below for Derby Network Driver.*
- b. Edit `dbConnectionURL` to replace `GLOBUS_LOCATION` with path to your installation. For example, if your `GLOBUS_LOCATION` is `/temp/globus`, then the property should look like `dbConnectionURL=jd-
bc:derby:/temp/globus/var/casDatabase`
- c. Comment out all other occurrences of these properties.

Now edit the server specific information as shown below:

Table 5.2. Test properties

<code>user1SubjectDN</code>	The DN of the user running the first set of tests.
<code>user2SubjectDN</code>	The DN of the user running the second set of tests. This DN has to be different from the value specified for <code>user1SubjectDN</code> . Note: Both tests can be run as the same user as long as the DN of the certificate being used to run the tests matches the value specified in <code>user1SubjectDN</code> . In this case, the value of <code>user2SubjectDN</code> can be set to an arbitrary string.
<code>maxAssertionLifetime</code>	Should match the value set in the service configuration as shown in <u>Configuration Information</u> .
<code>host</code>	Host on which the CAS service is running.
<code>port</code>	Port on which the CAS service is running.
<code>securityType</code>	This is an optional parameter indicating the security type to use. Can be set to <code>message</code> for Secure Message or <code>conversation</code> for Secure Conversation or <code>transport</code> for Secure Transport (the default configuration).
<code>protType</code>	This is an optional parameter indicating the protection type to use. Can be set to <code>signature</code> for integrity protection (the default configuration) or <code>encryption</code> for privacy protection.

2. Edit The `etc/globus_cas_unit_test/jndi-config.xml` to replace *GLOBUS_LOCATION* with the actual Globus Location of your install.

Steps for testing:

1. Run:

```
cd $GLOBUS_LOCATION
```

2. Source `$GLOBUS_LOCATION/etc/globus-devel-env.sh` or `$GLOBUS_LOCATION/etc/globus-devel-env.csh` or `$GLOBUS_LOCATION/etc/globus-devel-env.bat` as appropriate for your environment.

3. In the test properties file, set *user2SubjectDN* to the subject in your regular proxy. The following returns the appropriate string:

```
casadmin$ java org.globus.tools.CertInfo -subject -globus
```

4. Generate an independent proxy using the following command:

```
casadmin$ java org.globus.tools.ProxyInit -independent
```

5. Set the identity in the proxy generated from the above step as *user1SubjectDN* in the test properties file. The following command will return the relevant string:

```
casadmin$ java org.globus.tools.ProxyInfo -subject -globus
```

6. Start the container on the port and host configured in [CAS Test Properties](#).

7. The following command runs the tests for self permissions and sets up the database for a user with subjectDN *user2SubjectDN*:

```
casadmin$ ant -f share/globus_cas_unit_test/cas-test-build.xml user1TestService
```

8. To test as the second user, generate a proxy for the subject DN specified for the second user:

```
casadmin$ java org.globus.tools.ProxyInit
```

9. Now run the following:

```
casadmin$ ant -f share/globus_cas_unit_test/cas-test-build.xml user2TestService
```

10. Test reports are placed in `$GLOBUS_LOCATION/share/globus_cas_unit_test/cas-test-reports`.

11. After these tests, the CAS database needs to be reset and all entries need to be deleted.

```
ant -f share/globus_cas_unit_test/cas-test-build.xml testDatabase
```

 **Important**

The database bootstrap needs to be done again for the server to be ready to receive client requests.

Chapter 6. Example of CAS Server Administration

The following contains an example of administering the CAS server policies using the CAS administrative clients described. *FILLME: add olink to admin command line when its done.*

Alice, Bob and Carol are three members of a community who have set up a Community Authorization Service:

- Alice's role is primarily to administer the CAS server.
- Bob is an analyst who needs read access to much of the community data.
- Carol is a scientist who needs to be able to both read and write community data.

These examples show how:

1. Alice adds the users Bob and Carol to the CAS server.
2. Alice adds a FTP server with some data available to the community.
3. Alice adds permissions for the users using the CAS administration clients.

These examples assume the following:

- Alice has installed the CAS server and bootstrapped the database with herself as super user. Please refer to previous chapters in this guide for details on setting up the server and bootstrapping with data.
- Alice's nickname on the CAS server is *alice* and at bootstrap she has created a user group, *suGroup*, which has super user permissions on the database.
- The CAS service URL is `http://localhost:8080/wsrf/services/CASService`.
- For all commands listed below the environment variable `$GLOBUS_LOCATION` has been set to point to the Globus Toolkit installation and the commands are run from `$GLOBUS_LOCATION`.
- The environment variable `CAS_SERVER_URL` has been set to point to the CAS server URL, `http://localhost:8080/wsrf/services/CASService`.

1. Adding a user group

Since at the time of booting up the CAS server only the user group that has super user permissions on the CAS server is created, Alice will want to create another user group to which new users can be added and to which permissions on newly enrolled CAS entities may be given. This also eases the process of giving the same rights to many users. Given that there are two types of roles in the community she might want to create two groups, *analysts* and *scientists*.

Also, all permissions on the newly created group will be given to users of a particular user group. For example, Alice may want all users of the user group *analysts* to be able to manipulate the group.

To create a new user group Alice uses the *cas-group-admin* client. It requires a name for the new group being created, say *analysts*.

```
alice% cas-group-admin user create analysts analysts
```

This will create a user group *analysts* and give all users in that group the permission to manage the group (i.e add users, remove users and so on). She can similarly create a group called *scientists*.

2. Adding a trust anchor

Prior to adding Bob and Carol to the CAS server, Alice needs to ensure that the trust anchors for both have been added. If they share the trust anchor with Alice then this step can be skipped, since at bootstrap Alice's trust anchor would have been added to the database.

In our example Alice and Carol share a trust anchor different from Bob's. Therefore, Alice needs to add Bob's trust anchor by using the *cas-enroll* client with the *trustAnchor* option. She needs to provide details about the trust anchor such as the authentication method and authentication data used.

```
alice% cas-enroll trustAnchor analysts AbcTrust X509 "/C=US/O=some/CN=ABC CA"
```

The above will enroll a trust anchor with nickname *AbcTrust* that uses *X509* as its authentication method and has the DN specified in the command. The members of the *analysts* user group are given all rights on this object. This implies that any user who has this trust anchor is assumed to present credentials signed by this trust anchor.

3. Adding users

Now Alice can add Bob and Carol as users using the *cas-enroll* command with the *user* option. She needs to provide the user's subject DN and a reference to the trust anchor used by the user. As with any entity added to the CAS server, the admin needs to choose a user group whose members will have all permissions on that entity. In this example, Alice would like the members of the user group *suUser* to be able to manipulate the user entity *Bob*.

```
alice% cas-enroll user suUser bob "/O=Our Community/CN=Bob Foo" AbcTrust
```

Alice uses a similar command to add Carol to the CAS database.

4. Adding users to a user group

The CAS server allows rights to be assigned only to user groups and not to individual users. Hence, before Alice can assign rights to Bob or Carol, she needs to add them to some user group. She does this by using the **cas-group-add-entry** client with the *user* option to indicate she is adding to a user group. This client requires the group name and the nickname of the user who needs to be added. To add Bob to the *analysts* group, the command would be:

```
alice% cas-group-add-entry user analysts bob
```

If a user group *scientists* was created, Carol could similarly be added as a member.

5. Adding a new FTP server

Alice now has the community users in the database. The next step is to add some resources. Because the community currently has the FTP server *foo.bar.edu* available to it she will add it to the CAS database.

Each resource or object in the CAS server has a namespace associated with it that defines certain features. For example, it can define the comparison algorithm that is to be used when the object's name is compared. It may also define the base URL that should be prefixed to objects that belong to this namespace. In this case, Alice chooses to use the *FTP-DirectoryTree* namespace that is added to the CAS server at startup. She uses the *cas-enroll* client with the *object* option to add the FTP server to the CAS database:

```
alice% cas-enroll object suGroup ftp://foo.bar.edu/* FTPDirectoryTree
```

This command adds the FTP server as an object and gives all members of the *suGroup* rights to manipulate the object.

To be able to grant/revoke access on an individual directory, add an object for the directory. For example, if Alice would like to be able to manipulate the *data* directory on the server as a separate entity, the following command will add an object for that.

```
alice% cas-enroll object suGroup ftp://foo.bar.edu/data/* FTPDirectoryTree
```

6. Creating an object group

Alice suspects that the community will end up with more directories containing data on other servers that will have polices identical with the ones on the /data directory on foo.bar.edu. To manage this she is going to create an object group called *data* and assign foo.bar.edu/data to this group. This will allow her to grant rights on this group and easily add other directories to this group later.

To create a group called *data*, she uses the *cas-group-admin* client with the *group* and *create* options:

```
alice% cas-group-admin object create suGroup data
```

This creates an object group called *data* and the members of *suGroup* get all rights on this group and hence should be able to add/remove members, grant rights to add/delete from this group to others and also delete this group.

7. Adding members to an object group

Alice now can add foo.bar.edu/data to the *data* group. She can do this by using the *cas-group-add-entry* with the *object* option. To add the above object, *ftp://foo.bar.edu/data/** in the namespace *FooFTPNamespace*, to the object group *data* Alice uses the following command:

```
alice% cas-group-add-entry object data object FooFTPNamespace ftp://foo.bar.edu/data/*
```

In the above command:

- the first *object* refers to the group type.
- *data* is the name of the object group.
- the second *object* refers to the type of CAS entity that is being added as a member.
- the last two parameters define the namespace and the object that needs to be added.

8. Adding service types

Alice now needs to add information about the kinds of rights that can be granted for these objects. These are stored as *service types* and relevant actions are mapped to these service types.

In this scenario, the kind of service types that Alice should add would be *file*, *directory* and so on. To do so the *cas-enroll* client with the *serviceType* option may be used. To add a service type called *file* and give members of *suGroup* all rights on this service type Alice uses the following command.

```
alice% cas-enroll serviceType suGroup file
```

9. Adding action mappings

The relevant action mappings to the above mentioned service types would be *read*, *write* and so on. Alice needs to add these mappings to the database so that she can grant rights that allow a user to have *file/read* or *file/write* permissions on some object.

To add action mappings to a service type, she uses the **cas-action** client with the `add` option. The following command adds a mapping of action *read* to service type *file*.

```
alice% cas-action add file add
```

Similarly, she can add other mappings, like *write*, to this service type.

10. Granting permissions

Alice now has resources in the object group *data* and users in the user groups *analysts* and *scientists*. She now wants to grant permissions on the *data* group to the analysts and scientists, namely read permissions to the analysts and read and write permissions to the scientists.

To grant permissions Alice needs to use the **cas-rights-admin** with the `grant` option. To give read permissions to the analysts group Alice runs:

```
alice% cas-rights-admin grant analysts objectGroup data serviceAction file read
```

She similarly grants rights to *scientists* group.

Chapter 7. Example of using CAS Service for Web Service Policy Management

The CAS server can be used to store and manage policy for web services. The web service authorization decisions will require policy on a user accessing a particular operation on a said resource. Entities map to the CAS model as follows:

Table 7.1. Entity Mapping

User	A user group in CAS service with user as member.
Operation	A Service/Action entry is used to represent a web service operation. <ul style="list-style-type: none">• Service type is the OGSA-AuthZ URI for operation "http://www.gridforum.org/namespaces/2003/06/ogsa-authorization/saml/action/operation". This is added to the CAS server as an implicit object.• Action is the local part of the web service operation QName
WS Service/Resource	The EPR of resource represented as string. The EPR is converted to string using the algorithm specified in the OGSA AuthZ specification.

The next sections discuss a few different ways the CAS server can be used to manage web services access policy.

1. Example of using CAS Service as Authorization Service

This section describes using the CAS server as an authorization service to manage policy for web services. The sample service used here is the Counter Service distributed as a part of Java WS Core component. The CAS server needs to be bootstrapped with information about a super user and the sample assumes that the super user is the client. The methods on Counter Service that need to be protected need to be added as actions and the EPR of the created counter resource needs to be added as an object.

1. Configure the CAS service as described in [here](#)
2. Bootstrap the CAS service as described in [here](#). Ensure that the *user-subject* in the bootstrap properties file to be the DN of the client. For example, following is a sample bootstrap properties file

```
ta-name=ta
ta-authMethod=X509
ta-authData=/C=US/O=Globus/CN=Default CA
user-name=su
user-subject=/C=US/O=Globus Alliance/OU=User/CN=101497d3dcd.3dcd5aef
userGroupname=superUserGroup
```

3. For this example, configure the following sample descriptor as the service security descriptor for Secure Counter Service. It configures the relevant PDP to be used to leverage CAS for authorization.

Example of using CAS Service for Web Service Policy Management

- a. Download [sample security descriptor](#)¹ and save to `$GLOBUS_LOCATION/etc/globus_wsrf_core_samples_counter`.
 - b. Edit the descriptor file to have the identity of the authorization service as value of parameter `authzServiceIdentity`.
 - c. To configure it as descriptor, edit `$GLOBUS_LOCATION/etc/globus_wsrf_core_samples_counter/server-config.wsdd` and edit description for `SecureCounterService` to be as follows: `<parameter name="securityDescriptor" value="etc/globus_wsrf_core_samples_counter/counter-cas-security-config.xml" />`
4. Start up container with `-nosec` option.

```
bin/globus-start-contianer -nosec
```
 5. In another shell, set the `CAS_SERVER_URL` environment variable to point to the CAS service.

```
set CAS_SERVER_URL=http://localhost:8080/wsrf/services/CASService
```
 6. Also set the `CAS_SERVER_IDENTITY` environment variable to point to the identity of the CAS service..

```
set CAS_SERVER_IDENTITY="/C=US/O=Globus Alliance/OU=User/CN=101497d3dcd.3dcd5aef"
```
 7. Create a user group (`testUGp`) to give permission to access counter service.

```
bin\cas-group-admin.bat -m msg user create superUserGroup testUGp
```
 8. Add super user to the `testUGp`. The nick name for super user would have been specified in the bootstrap properties file. In this sample it is `su`.

```
bin\cas-group-add-entry.bat -m msg user testUGp su
```
 9. Add "createCounter" (an operation on counter service) as an action of the above service type.

```
bin\cas-action.bat -m msg add "http://www.gridforum.org/namespaces/2003/06/ogsa-authori
```
 10. Add "add" (an operation on counter resource) as an action of the above service type.

```
bin\cas-action.bat -m msg add "http://www.gridforum.org/namespaces/2003/06/ogsa-authori
```
 11. Add "destroy" (an operation on counter resource) as an action of the above service type.

```
bin\cas-action.bat -m msg add "http://www.gridforum.org/namespaces/2003/06/ogsa-authori
```
 12. The create counter operation is on the secure counter service (rather than any resource). The security descriptor will use the CAS service to authorize requests on the service. Hence policy needs to be written for the createCounter operation. First step is the enroll the `SecureCounterService` endpoint as an object in CAS.

¹ `../counter-cas-security-config.xml`

```
bin\cas-enroll.bat -m msg object testUGp http://192.168.1.101:8080/wsrf/services/Secure
```

13. In this command, policy for users in testUGp to invoke createCounter operation on the SecureCounterService endpoint.

```
bin\cas-rights-admin.bat -m msg grant testUGp object casDefaultNS http://192.168.1.101:
```

14. Now we create a new counter resource using the counter client and use the -b option to get the string representation of EPR. The string representation of EPR is used to add rights on the counter resource that was created in this command.

```
bin\counter-create.bat -s http://localhost:8080/wsrf/services/SecureCounterService -m m
```

15. Enroll the counter resource as an object. The object name will be the string representation of the EPR from previous step. Since we don't want any basename for this resource, use *casDefaultNS* as the namespace.

```
bin\cas-enroll.bat -m msg object testUGp "http://192.168.1.101:8080/wsrf/services/Secur
```

16. Grant rights for *testUGp* to be able to add to the counter resource.

```
bin\cas-rights-admin.bat -m msg grant testUGp object casDefaultNS "http://192.168.1.101
```

17. Attempt adding to the counter resource.

```
bin\counter-add.bat -e epr -m msg 10 -z none
```

18. Attempt destroying the resource. You will see an authorization exception.

```
bin\wsrf-destroy.bat -m msg -z one -e epr
```

2. Example of using CAS Service as Local PDP

This section describes using the CAS server as a PDP local to the container to manage authorization policy for web services. Unlike the previous section the authorization service interface of CAS is not used. But a PDP, `org.globus.cas.impl.LocalCasPdp`, distributed with the CAS server code base is used to contact a co-hosted CAS server for authorization decisions. This PDP uses Java calls on the CAS resource to obtain policy information and returns a decision of PERMIT or DENY.

For a service to use this feature, the authorization configuration at service/resoruce level should be the `LocalCasPdp`. To grant access to users, the users need to be added to the CAS database. This can be done either during bootstrap or during runtime. The resource owner can then facilitate fine grained, per operation authorization by storing appropriate permissions on the local CAS server.

Following is a list of steps that highlights how the `CasLocalPdp` can be sued:

- The test service used to as a part of this distribution to test this feature is used as the example service. The service consists of three methods, with following characteristics.

Table 7.2. Operation Details

Name	Purpose	Authentication Re-quired	Authorization Required	Invocation on ser-vice/resource
create	Create a new resource and return EPR. Initialize with integer value and an array of CAS user nickname allowed to access new resource.	Yes, any scheme	None	Service
getValue	Return current value of resource	No	No	Resource
add	Integer value as parameter to add to current value	Yes, any scheme	LocalCasPdp	Resource

- To configure the test service to use enforce authentication for create method, a security descriptor as shown below can be configured.

```
<serviceSecurityConfig xmlns="http://www.globus.org/security/descriptor/service">
  <auth-method>
    <GSISecureConversation/>
    <GSISecureMessage/>
    <GSISecureTransport/>
  </auth-method>
  <authzChain>
    <pdp>
      <interceptor name="none"/>
    </pdp>
  </authzChain>
</serviceSecurityConfig>
```

- To configure the test resource to enforce authentication and use LocalCasPdp for authorization, a security descriptor file can be written and loaded into the resource object as shown below

```
<serviceSecurityConfig xmlns="http://www.globus.org/security/descriptor/service">
  <auth-method>
    <GSISecureConversation/>
    <GSISecureMessage/>
    <GSISecureTransport/>
  </auth-method>
  <methodAuthentication>
    <method name="getValue">
      <auth-method>
        <none/>
      </auth-method>
    </method>
  </methodAuthentication>
  <authzChain>
    <pdp>
      <interceptor name="pdp:org.globus.cas.impl.LocalCasPdp"/>
    </pdp>
  </authzChain>
</serviceSecurityConfig>
```

```
</authzChain>  
</serviceSecurityConfig>
```

Sample Resource code, where resourceSecDesc is the path to the above file.

```
public class TestResource implements SecureResource, ResourceIdentifier {  
  
    ResourceSecurityDescriptor desc;  
  
    public TestResource(int val, String resourceSecDesc) throws Exception {  
        this.key = new Integer(hashCode());  
        this.value = val;  
        this.desc = new ResourceSecurityDescriptor(resourceSecDesc);  
    }  
  
    public ResourceSecurityDescriptor getSecurityDescriptor() {  
  
        return this.desc;  
    }  
}
```

- The owner of the service should exist in the CAS database with the same DN used to run the service. Also, a user group to add all new CAS objects created is required. For the purpose of this sample, we assume that "testUser" exists on the server and a user group "testUserGroup". Refer to [Section 2.3, "Bootstrapping the CAS database"](#) for details on how to bootstrap the CAS server.
- A Service Action group with all operations in the service should be added to the CAS service. This needs to be done only once and can be done in the resource home. This example shows how a new group "testActionGp" can be added.

```
String casUrl = "local://host:port/wsrf/services/CASService";  
Util util = new Util(casUrl);  
System.out.println("Creating service action group");  
util.createServiceActionGroup("testUserGroup", "testActionGp");
```

Note that local invocations can be used, since the CAS server is expected to be cohosted.

- All operation on the service should be added to the service/action group.

```
util.addOperation(new String[] { "getValue", "add"},  
                 "testActionGp");
```

- A User group per resource created can be added to the CAS server. This example uses the resource key as part of the group name.

```
String newUserGroupName = "objGp-" + key.toString();  
util.createUserGroup(userGroup, newUserGroupName);
```

Example of using CAS Service for Web Service Policy Management

- The EPR of resource should be added as a CAS object. The EPR should be converted to string representation and added with default namespace.

```
String eprAsString = EPRUtil.getEPRAsString(epr);  
// The user group on which all rights are granted for this object can  
// be the new user group created in previous step.  
util.addResourceObject(eprAsString, "testUserGroup");
```

- With all CAS objects bootstrapped, permission for the new user group to perform actions in new service/action group on resource should be added to CAS server. Now any new user who needs to be allowed access just needs to be added to CAS server and then added to user group for the said resource.

```
// set permission for new user group, resource EPR and then new action  
// group.  
util.grantPermission(newUserGroupName, eprAsString,  
                    actionGroupName);
```

- On a create operation, when the list of authorized users is passed as parameter, the user should be added to the newly created user group.

```
util.addUserToGroup(newUserGroupName, "authorizedUserNick");
```

- With the above set up, a create call can be made by anyone, provided some authentication is used. `getValue` can be invoked insecurely also. `add` will require that the CAS server have policy about the user.

Chapter 8. Security Considerations

1. Security Considerations for CAS

- The database username/password is stored in the service configuration file and the test properties file. Ensure correct permissions to protect the information.

Chapter 9. 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/4.2/4.2.1/common/javawscore/sample-container-log.txt>

Chapter 10. Troubleshooting

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

1. Error Messages

Table 10.1. WS A&A Authorization Framework Error Messages

Error Code	Definition
<p>org.globus.cas.impl.databaseAccess.CasDBException, connection refused</p>	<p>If the CAS service fails with following error:</p> <pre> faultCode: {http://schemas.xmlsoap.org/soap/envelope/}Server.userException faultSubcode: faultString: org.apache.commons.dbcp.DbcpException: Connection refused. Check that the hostname and port are correct and that the postmaster is accepting TC you need to troubleshoot the connection to the CAS database.</pre>
<p>CAS clients fail with org.globus.cas.impl.databaseAccess.CasDBException</p>	<p>If CAS clients fail with database permission exceptions similar to:</p> <pre> [Caused by: ERROR: permission denied for relation service_type_action]; nested exception is:org.globus.cas.impl.databaseAccess.CasDBException:</pre> <p>, then there is something wrong with user permissions on the database.</p> <p> Note</p> <p>This is a specific instance of an error for the relation <i>service_type_action</i>. This error could be raised on any rel</p>

2. Credential Troubleshooting

2.1. Credential Errors

The following are some common problems that may cause clients or servers to report that credentials are invalid:

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

Table 10.2. Credential Errors

Error Code	Definition	Possible Solutions
Your proxy credential may have expired	Your proxy credential may have expired.	Use <code>grid-proxy-info</code> to check whether the proxy credential has actually expired. If it has, generate a new proxy with <code>grid-proxy-init</code> .
The system clock on either the local or remote system is wrong.	This may cause the server or client to conclude that a credential has expired.	Check the system clocks on the local and remote system.
Your end-user certificate may have expired	Your end-user certificate may have expired	Use <code>grid-cert-info</code> to check your certificate's expiration date. If it has expired, follow your CA's procedures to get a new one.
The permissions may be wrong on your proxy file	If the permissions on your proxy file are too lax (for example, if others can read your proxy file), Globus Toolkit clients will not use that file to authenticate.	You can "fix" this problem by changing the permissions on the file or by destroying it (with <code>grid-proxy-destroy</code>) and creating a new one (with <code>grid-proxy-init</code>). Important: However, it is still possible that someone else has made a copy of that file during the time that the permissions were wrong. In that case, they will be able to impersonate you until the proxy file expires or your permissions or end-user certificate are revoked, whichever happens first.
The permissions may be wrong on your private key file	If the permissions on your end user certificate private key file are too lax (for example, if others can read the file), <code>grid-proxy-init</code> will refuse to create a proxy certificate.	You can "fix" this by changing the permissions on the private key file. Important: However, you will still have a much more serious problem: it is possible that someone has made a copy of your private key file. Although this file is encrypted, it is possible that someone will be able to decrypt the private key, at which point they will be able to impersonate you as long as your end user certificate is valid. You should contact your CA to have your end-user certificate revoked and get a new one.
The remote system may not trust your CA	The remote system may not trust your CA	Verify that the remote system is configured to trust the CA that issued your end-entity certificate. See Installing GT 4.2.1 for details.
You may not trust the remote system's CA	You may not trust the remote system's CA	Verify that your system is configured to trust the remote CA (or that your environment is set up to trust the remote CA). See Installing GT 4.2.1 for details.
There may be something wrong with the remote service's credentials	There may be something wrong with the remote service's credentials	It is sometimes difficult to distinguish between errors reported by the remote service regarding your credentials and errors reported by the client interface regarding the remote service's credentials. If you cannot find anything wrong with your credentials, check for the same conditions on the remote system (or ask a remote administrator to do so).

2.2. Some tools to validate certificate setup

2.2.1. grid-cert-diagnostics

The `grid-cert-diagnostics` program checks prints diagnostics about the user's certificates, and host security environment.

```
% grid-cert-diagnostics -p
```

2.2.2. Check that the user certificate is valid

```
openssl verify -CApath /etc/grid-security/certificates
-purpose sslclient ~/.globus/usercert.pem
```

2.2.3. Connect to the server using s_client

```
openssl s_client -ssl3 -cert ~/.globus/usercert.pem -key
~/.globus/userkey.pem -CApath /etc/grid-security/certificates
-connect <host:port>
```

Here `<host:port>` denotes the server and port you connect to.

If it prints an error and puts you back at the command prompt, then it typically means that the *server* has closed the connection, i.e. that the server was not happy with the client's certificate and verification. Check the SSL log on the server.

If the command "hangs" then it has actually opened a telnet style (but secure) socket, and you can "talk" to the server.

You should be able to scroll up and see the subject names of the server's verification chain:

```
depth=2 /DC=net/DC=ES/O=ESnet/OU=Certificate Authorities/CN=ESnet Root CA 1
verify return:1
depth=1 /DC=org/DC=DOEGrids/OU=Certificate Authorities/CN=DOEGrids CA 1
verify return:1
depth=0 /DC=org/DC=doegrids/OU=Services/CN=wiggum.mcs.anl.gov
verify return:1
```

In this case, there were no errors. Errors would give you an extra line next to the subject name of the certificate that caused the error.

2.2.4. Check that the server certificate is valid

Requires root login on server:

```
openssl verify -CApath /etc/grid-security/certificates -purpose sslserver
/etc/grid-security/hostcert.pem
```

Glossary

some terms not in the docs but wanted in glossary: [scheduler](#)

P

private key The private part of a key pair. Depending on the type of certificate the key corresponds to it may typically be found in `$HOME/.globus/userkey.pem` (for user certificates), `/etc/grid-security/hostkey.pem` (for host certificates) or `/etc/grid-security/<service>/<service>key.pem` (for service certificates).

For more information on possible private key locations see [this](#).

S

scheduler Term used to describe a job scheduler mechanism to which GRAM interfaces. It is a networked system for submitting, controlling, and monitoring the workload of batch jobs in one or more computers. The jobs or tasks are scheduled for execution at a time chosen by the subsystem according to an available policy and availability of resources. Popular job schedulers include Portable Batch System (PBS), Platform LSF, and IBM LoadLeveler.

service credentials The combination of a service certificate and its corresponding private key.

T

transport-level security Uses transport-level security (TLS) mechanisms.